

Deep Neural Networks in SAS® Enterprise Miner

By Russ Lavery

ABSTRACT:

Recent advances in algorithms and hardware (the GPU chip) have made it possible to build neural nets that are both deeper and wider than had been practical in the past. This paper explores the theory, and a bit of the practice, associated with the building of deep neural networks in SAS Enterprise Miner.

INTRODUCTION:

Neural networks got that name because of their similarity to the way neurons work in the human body. Any web research session on this subject returns mentions of neurons, so a small anatomy lesson might be worthwhile.

A cell is not a piece of undifferentiated jelly. Cells have structure and parts of cells have specific functions.

The cell has a nucleus that contains the DNA and parts that connect the cell body to other cells.

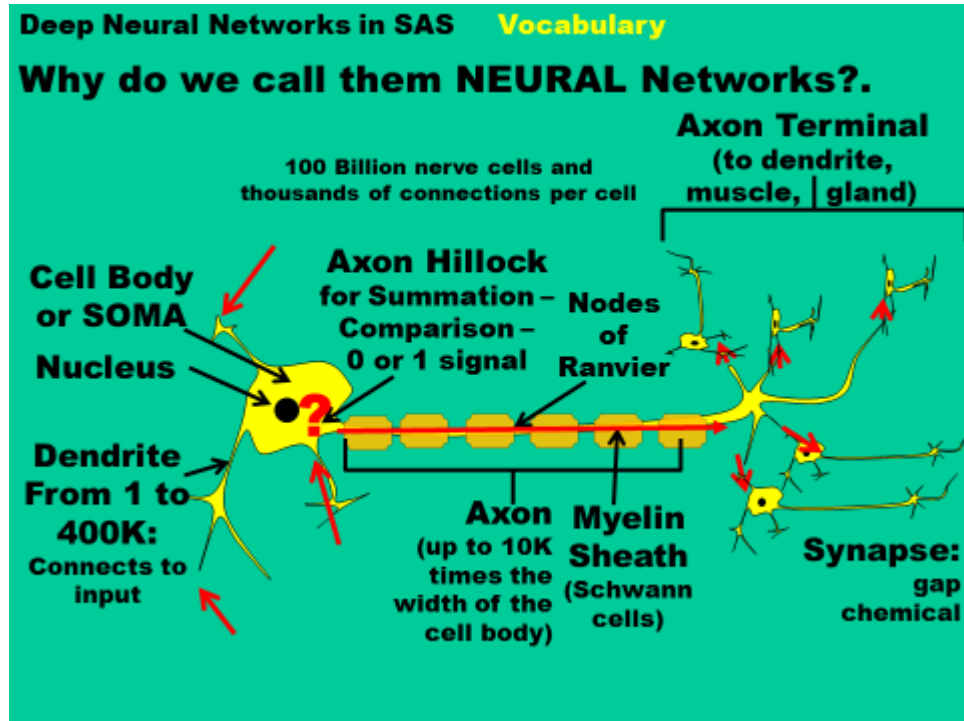


Figure 1

Dendrites are long stringy parts of the cell to take inputs. Axons send outputs to other cells. Your body is an incredibly deep neural network and one of your nerve cells can have hundreds of thousands of connections to other cells.

An input to the cell, maybe the feeling of a touch or sensing of a color in your eyes, comes in through a dendrite. Cells have many dendrites and can receive many simultaneous inputs. The individual inputs are summed, and "summed" is used in the same way that a mathematician would use the word, in a specialized part of the cell located adjacent to the start of the Axon. This specialized part of the cell, called the Axon Hillock, sums the different inputs and if the inputs exceeds some threshold the Axon Hillock sends an electrical signal down the Axon towards other cells (the cell "fires").

At the end of the Axon, the electrical signal is converted into a chemical signal that leaves the cell. A chemical signal bridges the gaps (the synapses) to other cells.

The important things to recognize are: 1) the huge numbers of connections between nerve cells and 2) the function of the Axon Hillock. It's job is to sum the different inputs, some of which might increase the chance of sending out a signal and some of which might decrease the chance of sending out a signal, and then to decide if it should send an electrical discharge down the Axon.

Figure 2 shows a small neural net but the characteristics of the small neural net are present in larger nets as well.

Nodes to the left are sometimes called "early" nodes.

A neural net can predict either binary or interval data and this net is trying to predict someone's weight from their sex, age and height.

A network has three types of nodes.

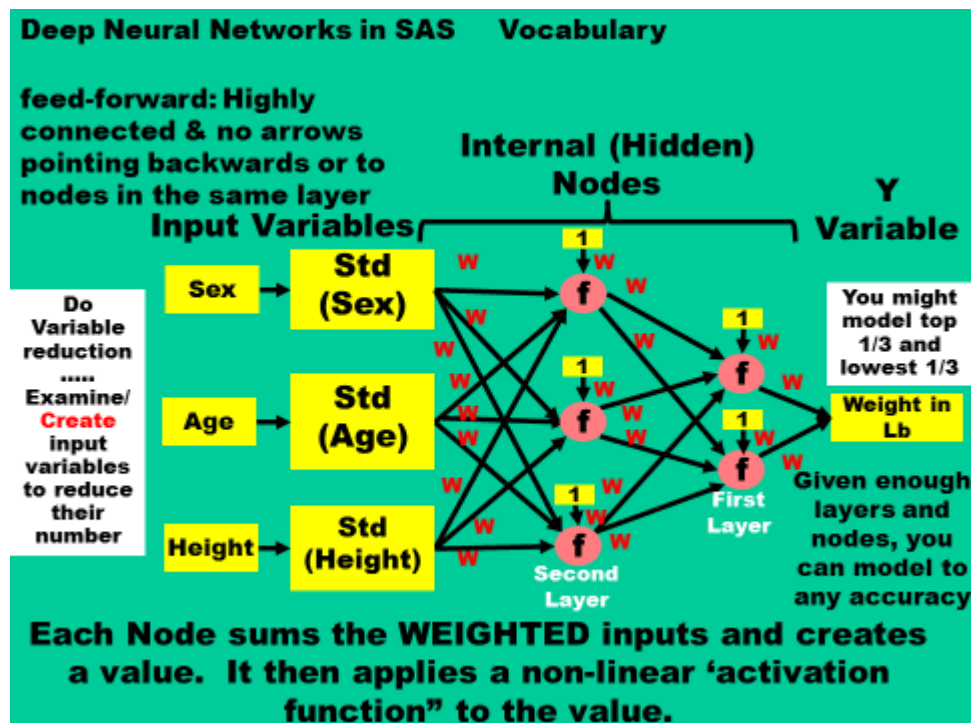


Figure 2

Networks have input nodes and there are three nodes in this input layer. Networks have internal (often called hidden) nodes and layers. This net has two hidden/internal layers. The first layer has three nodes and the second layer has two nodes. Networks have an output layer and this network has one node in the output layer.

The network in figure 2 is a feedforward node. Each node in a layer to the left is connected to every node in the layer immediately to its right. There are no connections backwards between nodes, so no arrows point to the left. Finally there are no connections between nodes in the same layer.

Inside each node is a function (represented by the letter F in the circles). These functions are referred to as activation functions, transfer functions or simply transforms. The functions are usually nonlinear and common ones are linear, logistic, hyperbolic tangent and Gaussian. The fact that these transfer functions are usually non-linear makes the whole neural network non-linear. A neural network has the ability to separate groups (and that is what predicting a binary Y is doing) with a boundary that is very curved and irregular.

The basic process is to take the values of a person's sex, age and height and enter them into the input nodes. The input variables are often standardized to remove the effects of different measurement units. The values of sex, age and height are multiplied by the weights (the red Ws) and the result is passed on to the internal nodes. Each internal node receives many inputs. Some people think of neural network weights as being similar to the beta coefficients in a regression. Neural net weights, like regression beta values, are measures of how much impact an X variable has on the Y variable. Arrows indicate how values are combined. At the right side of the network, the sum of weighted inputs (after going through all the nodes) is compared to a known Y value and an error is calculated. The back propagation algorithm then takes the derivative of the error with respect to each of the weights and uses that derivative to adjust the weights to produce a smaller error.

Think of each person's sex, age and height entering this network - the three variables enter simultaneously - one person at a time. The weights are, for the first person read, set to random numbers and they produce large errors. After each observation is processed, the weights are adjusted to reduce the error and after many (often several thousands) subjects are processed, the weights can predict the Y value with small error. A second pass is needed, using the final weights, to score all the observations.

If a reader looks at the top node in the first internal layer s/he can see that it has inputs from sex, age and height as well as from a one (coming from a yellow box). The one is called a bias term and it is used to adjust the summed values from the input node so that the result, after adding in the weighed bias, has a value that does not "overload" the transform function. Overloading is most easily explained by thinking of the activation function as being a

Gaussian transform – a bell shaped transform. The input to the activation function is the Z value (the summed weighted inputs from previous nodes) for the Gaussian and the output of the transform is the height of the bell above that value of Z. If Z is +3, the transform returns a value close to zero. If Z is +8, the transform also returns a value close to zero. After a Z value exceeds a certain absolute value, the transform returns, for practical purposes, the same value and is both “overloaded” and no longer sensitive to changes in Z. The bias is used to “move” the value of Z back to a value where the transform function is more sensitive to changes in Z.

Inside the node, the inputs are summed and then pushed through the function in the middle of the node to produce an output value for the node. I tend to think of each node as holding two numbers: an input number and an output number. An input number is the weighted sum of all of the values coming in from the left and the weighted bias. An output value is the one number that is a result of applying the transform function (also called activation function) to the summed weighted input values (the input number).

In early research, the activation functions were often just step functions. If the summed weighted input values was not above a certain level (a cutoff number), no value (or maybe a zero) was passed on to nodes to the right. Now the nodes use smooth S shaped functions (or maybe bell-shaped) and they always pass on some value to nodes to the right – though the value may be small.

Given enough nodes, and layers, you can model any data set to any desired level of accuracy – though it might take a very long time if the data set is large.

If you feed, into the network, an X variable that has no predictive power (e.g. a code for “blue eyes” vs “not blue eyes” in our problem of predicting weight) the neural net will eventually assign weights of zero to eye color. If you have enough data, and enough time to wait for the algorithm to run, a neural net will remove non-predicting variables by setting their weights to zero. However including a lot of silly variables as inputs will make the neural net run longer and possibly increase the chance of it finding a local Optima.

Figure three shows some of the activation functions that researchers use.

Linear is often used to connect the last hidden layer to the output layer.

Hyperbolic tangent and Gaussian activations are also commonly used in other parts of the network.

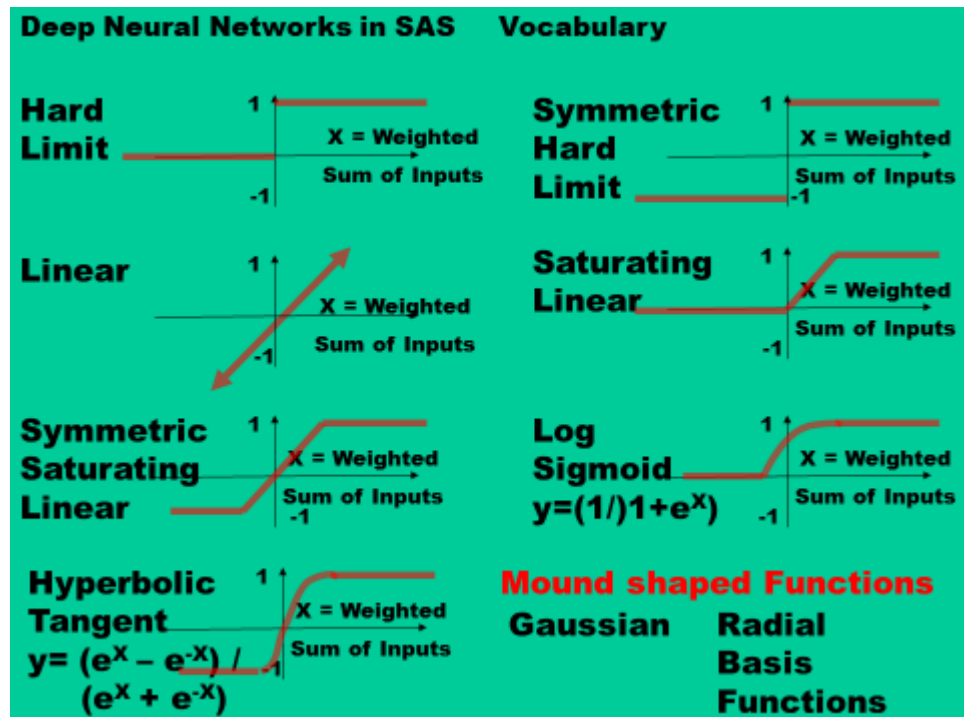


Figure 3

Figure 4 facilitates a discussion of why non-linear functions are so commonly used.

Biologists think that frogs' brains contain two neural networks to help it find flies to eat.

One network matches the size of the object to the size of an ideal fly. The other network matches the "flying behavior" to that of an ideal fly.

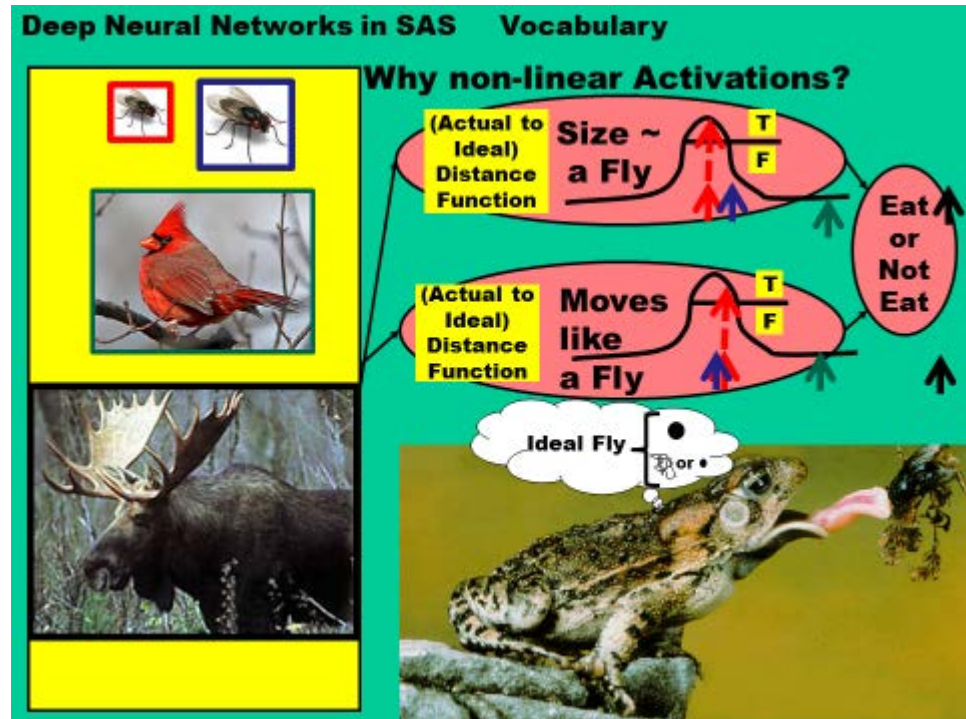


Figure 4

This paper will next discuss how a frog might use a Gaussian function to evaluate several potential meals. The choices are: a small fly (red border and arrow), a large fly (blue border and arrow), a bird (green border and arrow) and a moose (black border and arrow). The activation functions are mound shaped and the X value (horizontal value) generated by each object are "object distance from ideal". Close to the ideal points, the function returns a large value (it "fires"). There is a cut-off value, shown as a horizontal line on the function, at which point the frog decides if "eat=True" or "eat=False" (or "activate" vs "not activate", "fire" vs "not fire").

For the small fly, both the size and flying behavior are close to the ideal (see red arrows). Both networks return a large, "above the cut-off", value and "lunch is served". For the large fly, the size is a bit off-putting, though the flying behavior is close to the ideal (see blue arrows). Both networks return large values and the frog would likely attack. Because of the non-linear shape of the activation function, the networks are sensitive to small changes in the area of the "ideal".

For the bird, the size and behavior are both wrong (see green arrows) and the networks return two low values. For the moose, both the size and behavior are very wrong (see black arrows) and the networks return two low values. Because of the non-linear shape of the activation functions, the values returned for the bird and moose are similar. This makes sense because once the frog had decided that an object is "not lunch" it does not need to make fine evaluations of "how much not lunch" an object might be. Because of the shape of the non-linear activation function, the networks are NOT-sensitive to small changes far from the "ideal".

Figure 3 shows a larger net, though far from being a very large net these days. You can see there are lots of connections between lots of nodes.

Neural nets are used in digital cameras to identify faces of people in a picture.

Much exciting work is being done in visual recognition using neural networks.

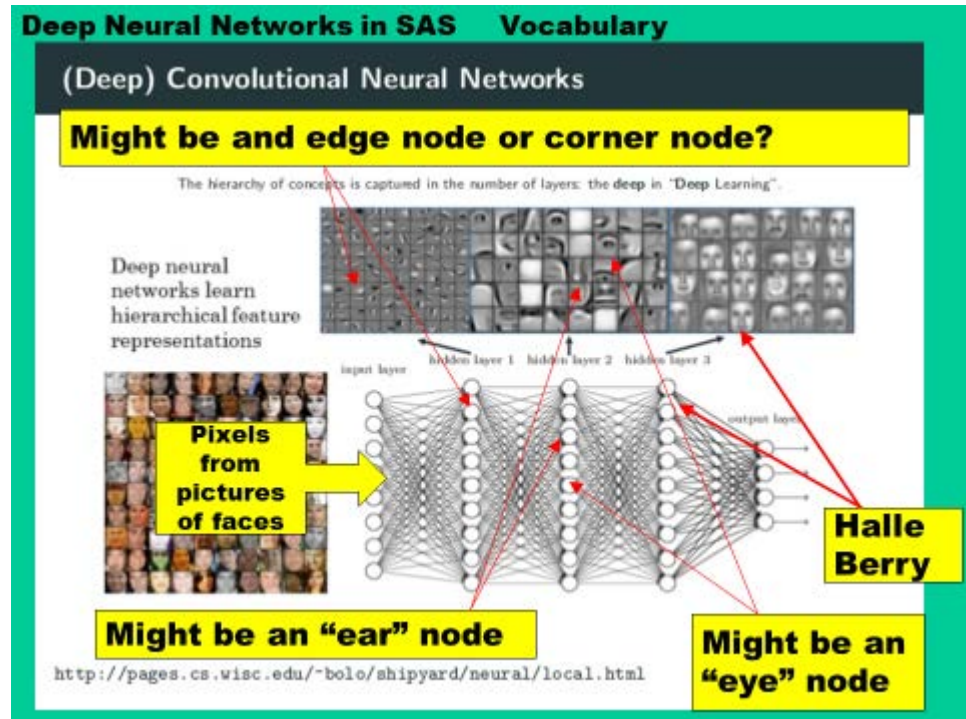


Figure 3

There was, and to some extent still is, a criticism of deep neural nets that they are black boxes – that the results can be very good but no one can understand how the results are created. Recent research has made that statement less true. Visual recognition research has allowed people to peek inside of neural nets and discover some exciting findings. This paper will discuss the internal processes of neural networks using pictures as the research issue.

It seems that early layers in the net identify basic visual building blocks; like edges going from light-to-dark or dark-to-light. Nodes farther to the right, in the net, can create higher level abstractions. Nodes in the middle of a neural net might identify parts of faces, like ears or noses. Nodes to the far right of the neural net can reconstruct faces and even recognize people.

WAYS TO USE SAS TO CREATE NEURAL NETWORKS:

SAS Enterprise Miner has four ways to do neural nets.

DMNeural uses bucketed principal components as X variables and can predict a binary or interval Y. HPNeural is designed as a high performance modeling tool. It will access memory across multiple cores and multiple computer nodes. It is not good for deep neural nets because it does not provide protection against the problem of vanishing or exploding gradients. Auto Neural conducts limited searches to help you find a better network architecture. It will try different numbers of layers and nodes as well as different activation functions.

Neural network is the SAS work horse for doing neural nets and will process a deep neural network. It provides the most control and most power of the choices that SAS provides. In order to do a deep neural net you must have Enterprise Miner installed, but it is easy to code a PROC Neural in the SAS display manager once you have installed Enterprise Miner.

A NEURAL NET PROCESS:

Good Neural Network results are the result of a multi-step (multi-node?) process and this paper will examine some of the other steps. Good neural network results come from a process and the work done before the neural net is important. Steps in a good process might be:

Sampling can reduce the time to train a neural net and quick run times are always desirable. A researcher must balance the desire for quick run times with the fact that training a complex neural network to do a complex task requires lots of training data. To some extent, the quality of the results depends on the quality, and amount, of the training data.

Programmers usually want to create partitioned data sets to allow SAS to *automatically* report on how well the neural net performs on data that is different from the training data.

Consulting with business experts, and doing exploratory modeling, can reduce the number of variables that must be feed into the neural net. Often having fewer, and higher quality, input variables reduces training time and improves the results.

An analyst might want to impute missing values or transform data before passing it into a neural net. Neural nets are highly non— linear but transforms of the X variables can reduce training time.

A programmer might want to remove outliers because they can reduce model accuracy.

A neural net usually needs a data mining database (DMDB) catalog entry and a researcher might need to run PROC DMDB before her neural net will run.

Finally, in a neural net project, an analyst might also want to use other modeling nodes. It might be that the neural net is not the best technique for any particular use case.

A “COCKTAIL PARTY” HISTORY OF NEURAL NETWORKS

The seminal article for neural nets was written by Donald Hebb in 1949. He wrote, about neurons in the body and said, “when an Axon of cell A is near enough to excite cell B, and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased.” Hebb was hypothesizing that “neurons that fire together wire together” and his article was the start of an explanation of how neurons are involved in learning and memory.

Efforts to make computers work like human cells started soon after Hebb’s article. People were doing research using computers and electrical circuits in the 1950s. In 1963 Vapnik and Chervonenkis discovered the idea of the support vector machine.

A book, in 1963, threw a major monkey wrench into neural net research. Papert and Minsk, in their book titled “Perceptrons”, demonstrated that a single node can classify successfully only if the Y classes in the data are linearly separable. They also proved that a single layer perceptron could not learn the logical XOR function. The inability to learn the XOR function was seen as a major, and general, flaw in neural networks and machine learning. Research interest plummeted.

Interest was revived when, in 1974, Paul Werbos invented a training method called backward propagation. This allowed for the creation of multi-node and multi-layer neural nets, though it ran into a problem called “the vanishing gradient” when applied to large nets.

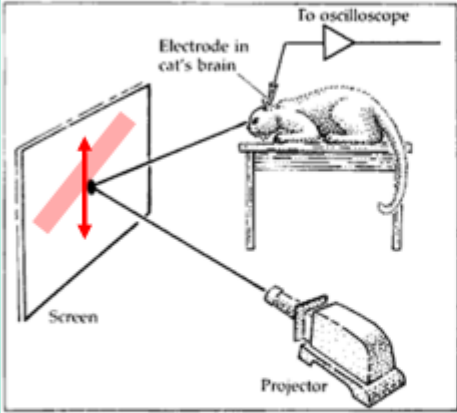
Restricted Boltzmann machines were invented by Smolensky in 1986 but became important in the early 2000s as Geoffry Hinton applied them to machine learning and the creation of Deep Neural Networks.

In 1981 Hubel and Wiesel won a Nobel Prize for work on neuronal activities and vision. They had embedded an electrode in a cat brain and struggled to measure some sort of neuronal activity driven by pictures projected in front of the cat.

Their first signal came when the cat saw a straight line as they changed slides.

Deep Neural Networks in SAS History
The cat who saw the edge
Hubel & Wiesel : Nobel Prize in Physiology and Medicine in 1981

The research showed researchers could map visual activity in the brain
Struggled, for a long time, to find neuronal activity in a cat brain
First discovery was just luck
Showed pictures of spots
Pictures of women from magazines
Jumped around and waved their arms



When they changed a slide, the neuron fired - it had been trained to see an edge

<https://www.youtube.com/watch?v=IOHayh06LJ4>
 Hubel & Wiesel - LGN Neuron <https://www.youtube.com/watch?v=9qg9-nBJUT>
 Visual Cortex Cell Recordings <https://www.youtube.com/watch?v=VPQAtkxn3tY>

Figure 4

It turns out that lines, or edges, might be important for both animal vision and for computer vision. In figure 3 we can see that early layers in the artificial neural net seem to be detecting lines of varying types.

Research into vision is particularly amenable to discovering what's going on in the inner layers of the neural net. This paper will discuss some image recognition tasks, and logic, as a way of building familiarity with the neural net internal process.

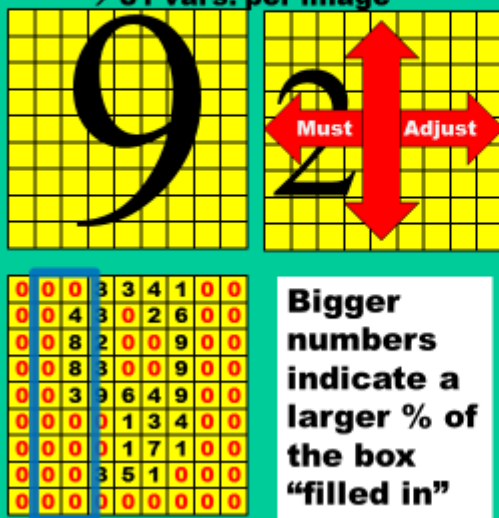
In figure 5 we get some idea of how pictures are coded.

In this figure we see how early number recognition research was coded.

Numbers were written on an input area that had been divided into a 9 x 9 grid (one can obtain better results if coding is at a pixel level but this is hard to put on a ppt).

Each cell was coded as to dark vs light.

Deep Neural Networks in SAS Theory
How to code a picture hand written number into a vector of observations
Create 9x9 Blocks (score white to black)
→ 81 vars. per image



The input vector is 81 rows by 1 column... hard to fit on slide

This is →

Just PART OF the input vector for the "9"

0	0	0	3	3	4	1	0	0
0	0	4	3	0	2	6	0	0
0	0	8	2	0	0	9	0	0
0	0	8	3	0	0	9	0	0
0	0	3	9	6	4	9	0	0
0	0	0	0	1	3	4	0	0
0	0	0	0	1	7	1	0	0
0	0	0	3	5	1	0	0	0
0	0	0	0	0	0	0	0	0

Bigger numbers indicate a larger % of the box "filled in"

Figure 5

The 81 cells were arranged in an 81 x 1 input vector and that input vector could be sent to a neural net with 81 input nodes. A little thought, and a peek at the number "2" in the middle of the slide, will lead a reader to recognize that numbers might need to be adjusted for position, and size, before being put into the neural net input vector. Above is a basic process for number recognition. State-of-the-art vision technology, attempting to recognize people and objects in photographs, will input each pixel level - coded for multiple colors - and the input vector will be much larger.

Early nodes in the network assemble the pixels into things like: vertical edges, horizontal edges (see right), angles or types of circles. Later nodes will assemble those edges into numbers.

As a warning the neural net here would not be able to input an 81 variable input vector.

With only four output nodes it would also be unable to correctly identify 10 digits.

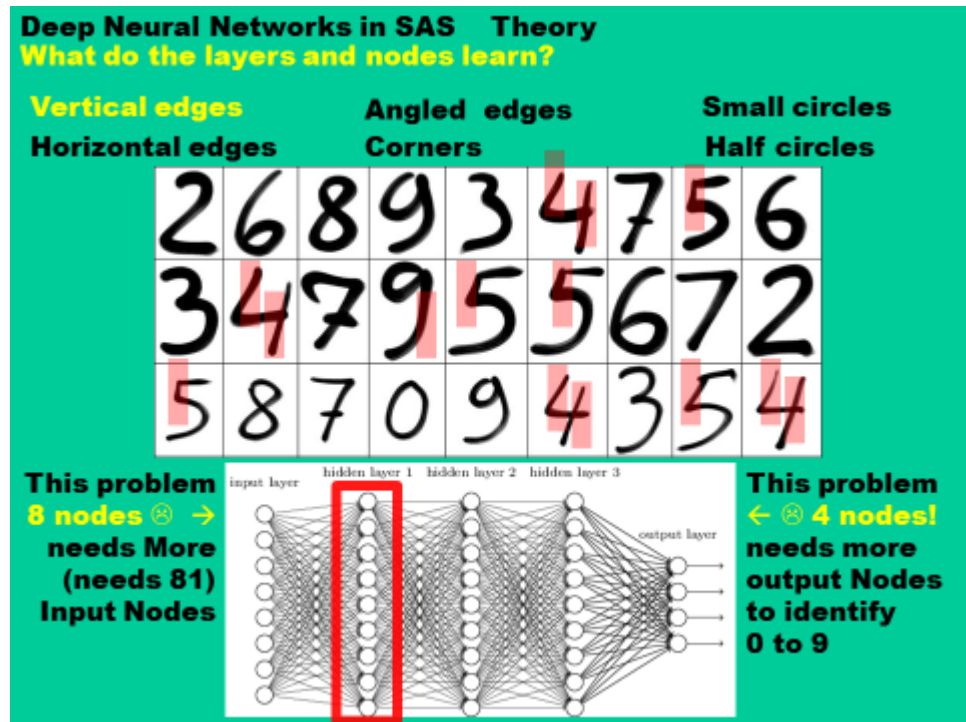


Figure 6

The technologies used to recognize digits can be transferred into more complicated problems like recognizing faces.

Parts of faces can be decomposed into simpler geometric shapes and the shapes built up into things like eyes and noses and mouths.

Here we see partial circles being recognized in numbers and geometric shapes being "found" on photographs.

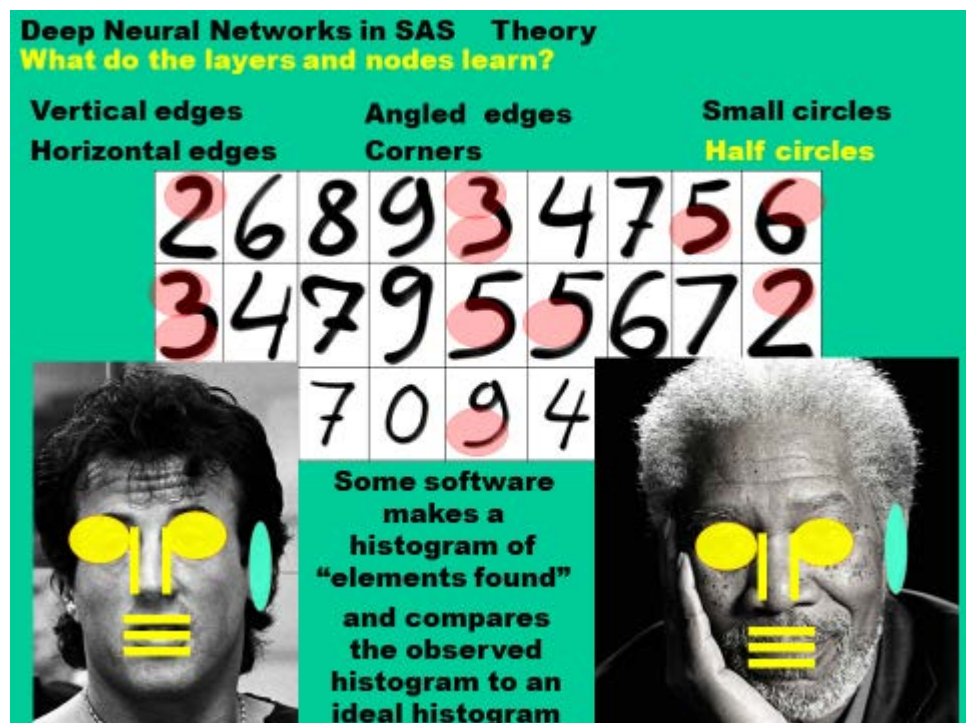


Figure 7

Some early software made histograms of “elements found” and compared the observed histogram frequency to some ideal histogram. You can imagine that the software said, “ two cat ears, fur, two eyes with slits, one long wavy tail and about twenty-four whiskers matches the histogram frequency for cat”. Some flexibility is required because, as you can see from these pictures of movie stars above, not all pictures show all the components associated with a type of animal. Both of these, professionally photographed, movie stars appear to have only one ear.

ALGORITHMS USED IN DEEP NEURAL NETWORKS:

A fairly deep dive into the algorithms involved in neural nets will help make some of the vocabulary more clear. Some detailed, and worked out examples, will be very helpful to anyone studying this field.

This example is taken from “A Step by Step Backpropagation Example” by Matt Mazur and can be found at: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example>. Full details are in the appendix of this paper.

In figure 8 we see some of the notation that we will use later on in the paper and in the appendix.

This is a small neural net with two input nodes, two hidden nodes and two output nodes.

It performs a binary classification and will assign probabilities of being a “top” or “bottom”.

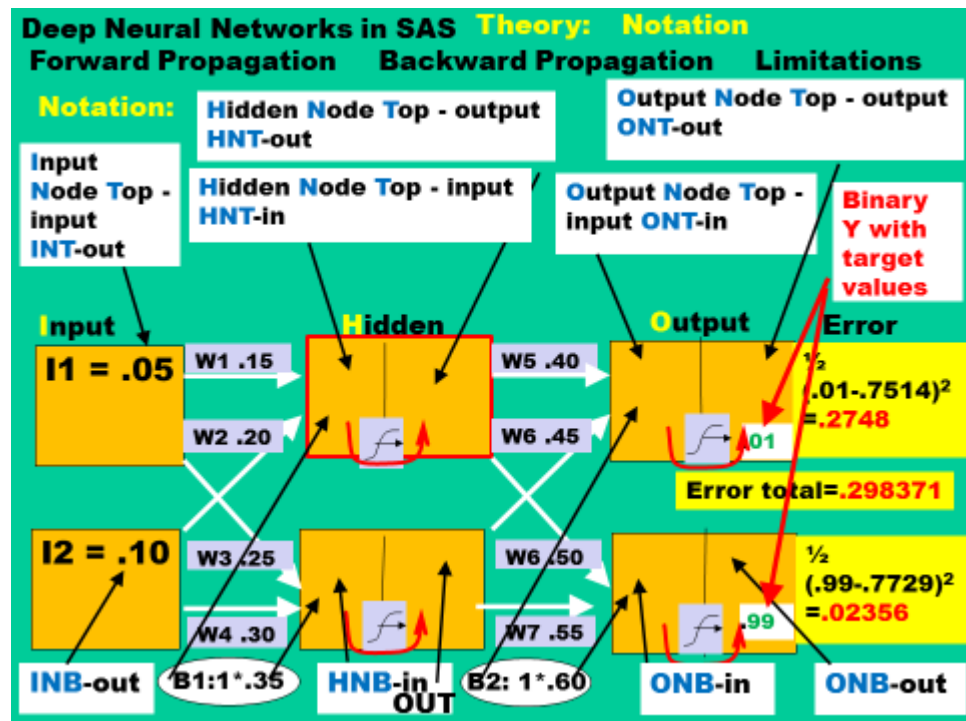


Figure 8

For the observation currently being processed, input node one has a value of .05 and input node two has a value of .10. Please remember that nodes, in other layers, have an input value, an activation function and an output value and this leads to our naming convention. HNT-in stands for hidden node top path input. HNT-out stands for hidden node top path output. In this neural net, since the output nodes have an activation function, output nodes also contain two values.

B1 and B2, in the white ovals, are bias variables. The weights of the bias variables, in any real neural net, will also be trained to minimize the prediction error. We will not do that training in this example.

Figure 9 shows forward propagation.

Initially all the weights are assigned randomly to numbers close to zero and the numbers in this slide are not unreasonable.

Forward Prop starts by taking the input values and multiplying them by their weights and sending them onto the next node to the right.

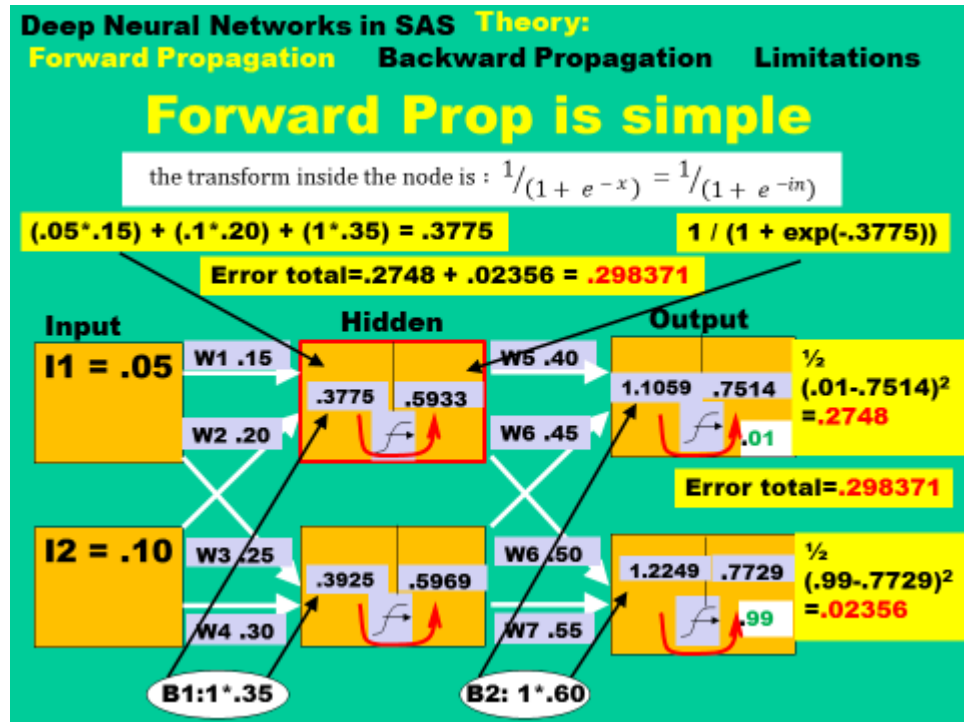


Figure 9

The .3775 in HNT-in is the sum of the weighted inputs to that node. The calculation for the .3775 is shown in a yellow box in figure 9. The transform used inside all of these nodes is shown in the white box on figure 9 and is $1 / (1 + \exp(-x))$. HNT-out is: $1 / (1 + \exp(-.3775))$. If the process is repeated for all of the other nodes a reader can recreate the input values and output values of the hidden and output nodes.

The observation also has an observed probability (this number is the result of a human rating and was contained in the training data file) of being a “top” of .01. This observation has a probability of being a “bottom” of .99. The predicted value for being a top is .7514 in the error component for top .2748. A similar process allows us to calculate the error associated with bottom. If we sum the two errors we get the total error- for this observation and for these weight values.

Now we now want to adjust the weights, in a very logical manner, so as to reduce the total error.

A neural network used to start with randomly assigned, near-zero, weights. The algorithm would read an observation and adjust the weights. Prediction errors for the first several thousand observations would be large, but that was not important. What was important was the final rules after thousands of “training cycles”. In a second step, the whole data set could be “scored” by applying the derived rules. Neural networks can be sensitive to starting weights and, now, there are several techniques that can replace, and improve on, a “random assignment of starting weights”,

Adjusting the weights is called “training the neural network” and often uses a process called “back propagation” (AKA back prop). Back propagation involves taking the partial derivatives of the error with respect to each of the weights. This involves using a calculus technique called the chain rule. In the paper itself, we will not show all of the steps because several steps are repetitive. However, in the appendix we will paste, into the paper, all of the steps for a backward propagation so that an interested reader can reproduce the work. It is hoped that the example in the appendix is a valuable part of the paper.

The paper will start by training weight five (W5), the weight in the gold box. W5 affects ONT-in and, through the activation function, it also affects ONT-out and thereby error.

The white box in figure 10 shows the chain of derivatives we must follow/calculate.

As you can see in the white box, we must calculate three terms.

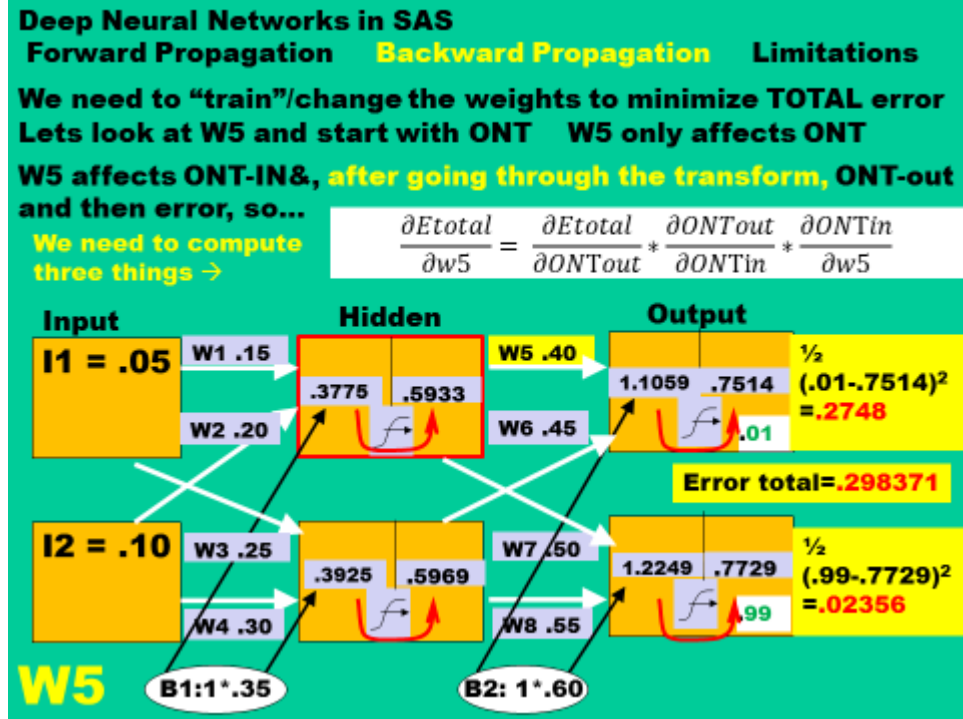


Figure 10

Figure 11 shows the calculation of the first term in the equation on Figure 10. We calculate the partial derivative of the total error with respect to ONT - out.

The value of this term is .7414.

Note that changing the value of W5 only affects one error term - the top error.

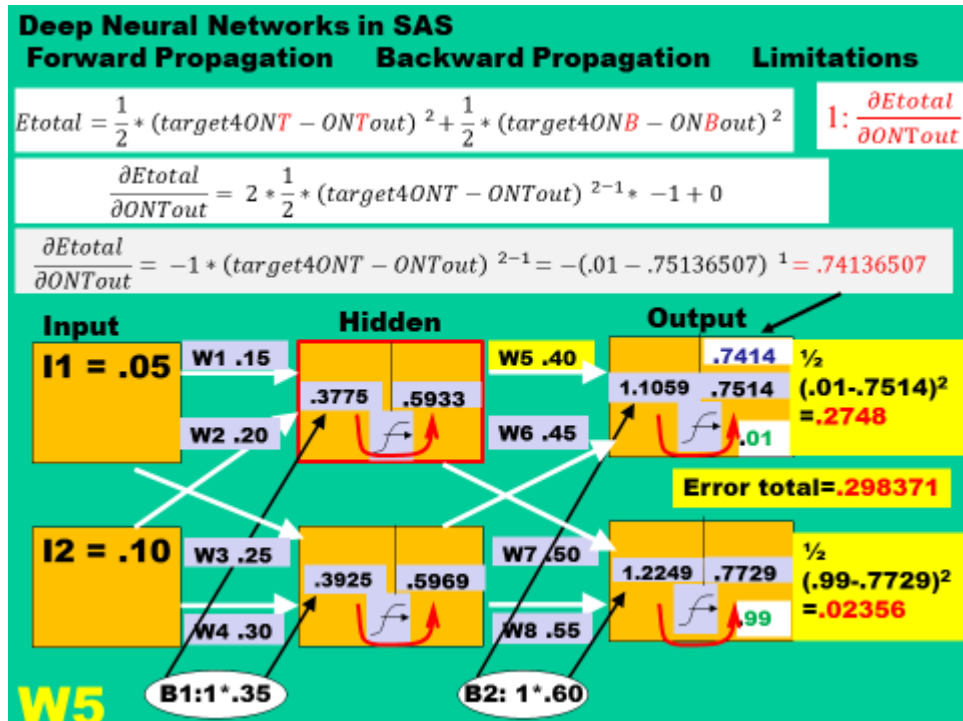


Figure 11

Figure 12 shows the calculation of the second term of the equation. In this step we move “our number” “back through” the transform – back through the activation function.

The second term of the equation has the value .1868.

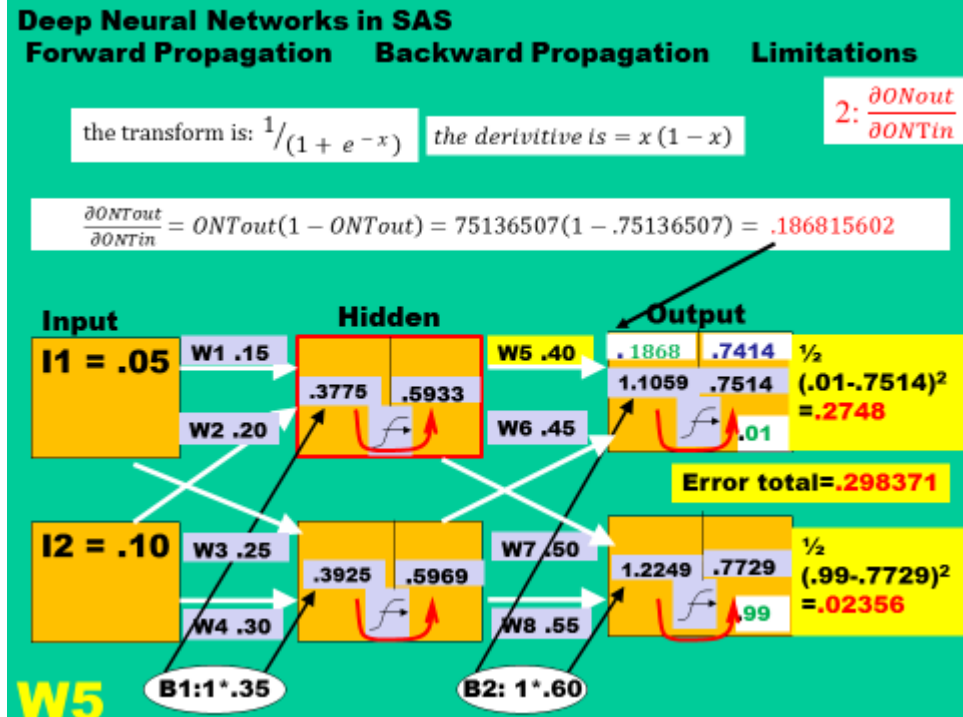


Figure 12

Figure 13 shows the calculation of the third required term and, in the large white box, a reader sees the multiplication of the three terms together.

This calculates that the partial derivative of the total error with respect to W5 is .082167.

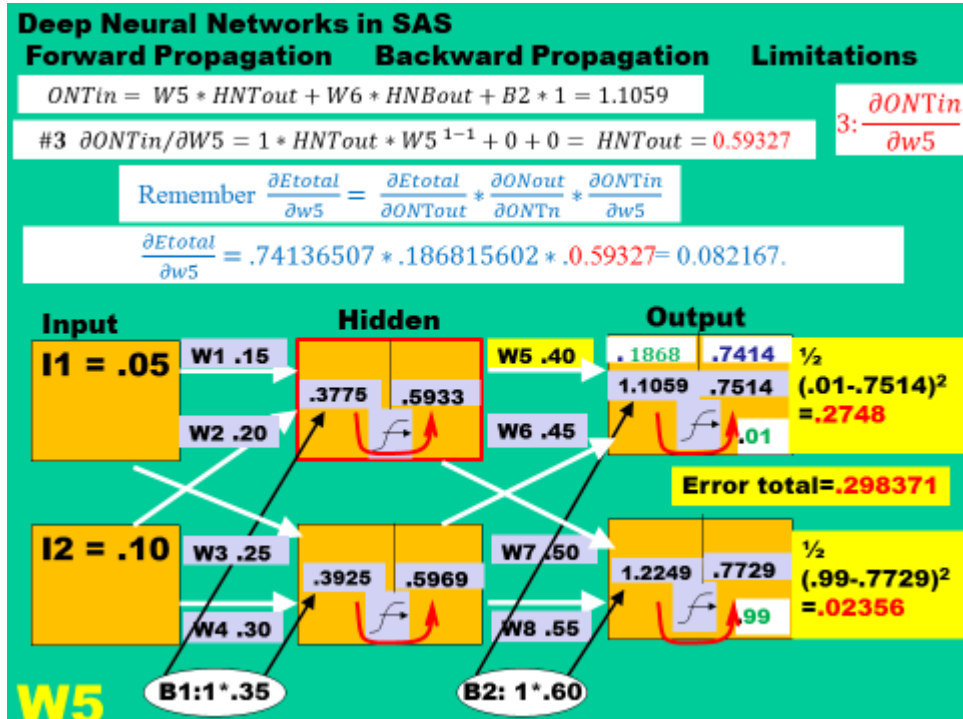


Figure 13

Figure 14 shows the final adjustment to W5. Our formula suggests that we should adjust W5 by .082167041 but this is likely to be too strong an adjustment.

An adjustment this large is likely to cause the algorithm to overshoot the optimal and create a situation where the algorithm oscillates wildly.

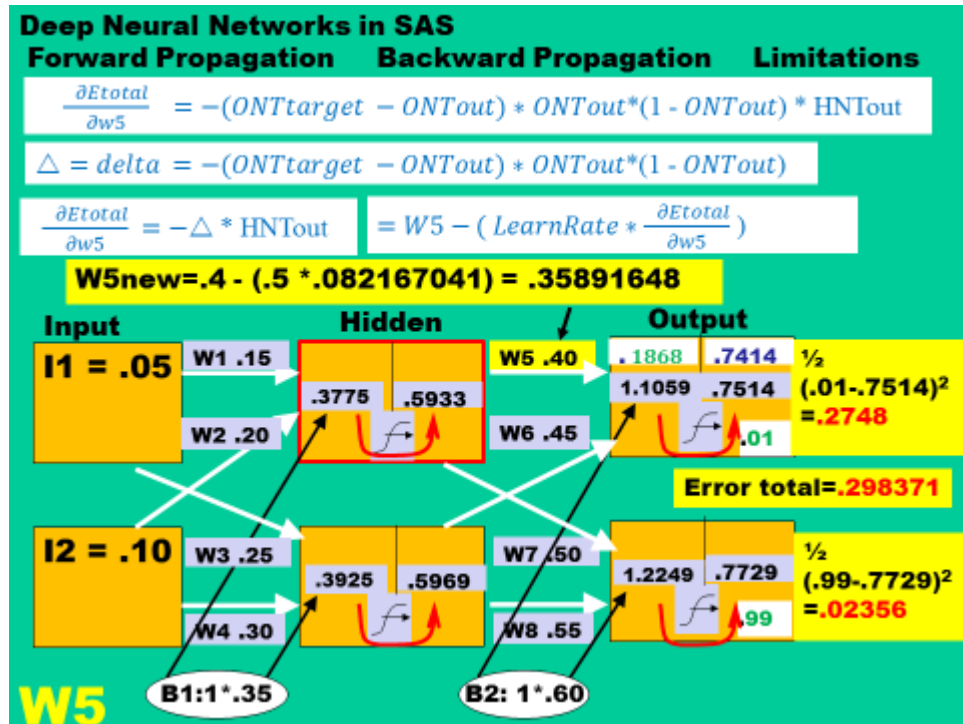


Figure 14

To avoid oscillation, back prop applies what is called a learning factor – the .5 in the equation. Because we set the learning factor to .5, back prop applies just half of the adjustment that our formula suggests. This smaller adjustment will result in the algorithm taking more steps to reach the optimal solution but software designers were willing to pay that price to decrease the chance of unstable oscillations. Enterprise Miner allows a user to change the value of the learning parameter.

Informally speaking, the .1868 and the .7414 are “characteristics” of the top output node. If a formula requires a partial through output node top, these numbers do not need to be recalculated. Therefore; when adjusting W6, most of the work is already done. Details of adjusting W6 are left to the appendix.

The training for W7 and W8 proceeds with steps similar to those in the example shown for W5. Details of those adjustments are left to the appendix as well. Please note that adjusting weights W5 to W10 would only affect one of the two error terms.

Adjusting the weights for W1, W2, W3 and W4 will be a different process from that of adjusting the weights W5 to W8. The process of adjusting W1, W2, W3 and W4 will be more complicated than adjusting W5 to W8 because changing W1, W2, W3 or W4 affects both of the error terms.

Figure 15 shows how changing W1 affects both of the error terms. The top white box shows that the partial derivative formula is very similar to the one we used before.

We want to be sure to follow the yellow arrow downward to see how total error has two error components; top and bottom.

The two error components will have make the resulting process a bit more complicated. It will have two parts.

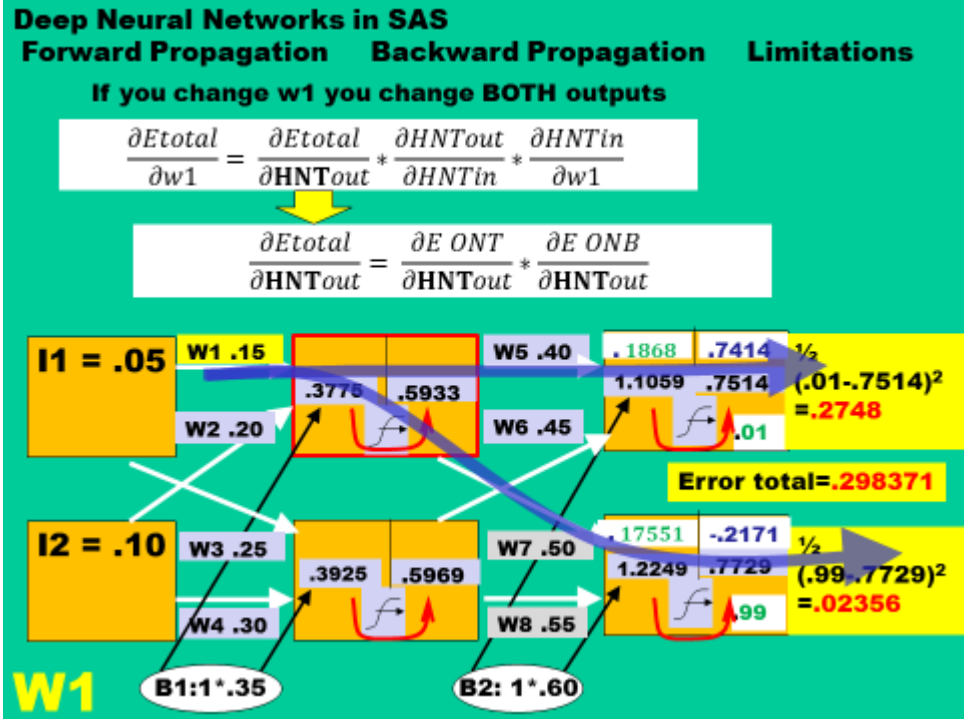


Figure 15

The new process for adjusting weights will have two components – one that recognizes the effect of a weight on the top error and one that recognizes the effect of changing a weight on the bottom error.

Figure 16 is intended to emphasize the three-step process that we must again follow as we adjust weights.

Fortunately, much work has been done.

Numbers that were described as “characteristics of the output nodes” will be used in these new formulas.

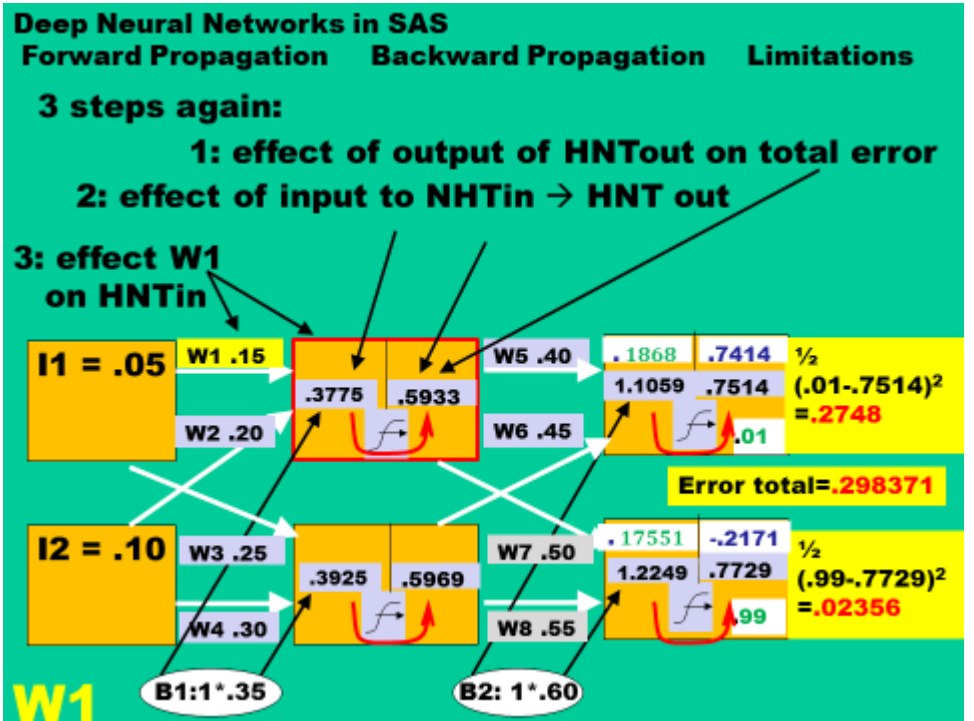


Figure 16

Figure 17 emphasizes that there are two error terms (IONT and ONB) that must be accounted for as we take the partial derivative through HNT.

The number coming back to the output side of HNT is .0364. To take that partial derivative through the transform, in reverse order, results in the number .241300700

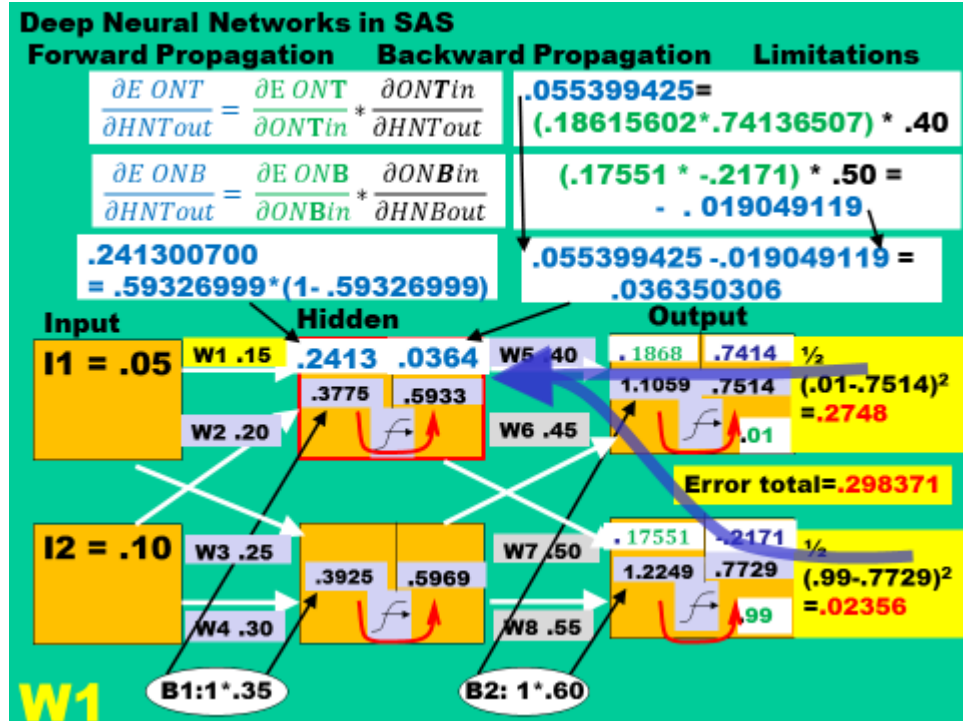


Figure 17

Figure 18 shows the three-part formula in mathematical terms (as partial derivatives) and also in numerical form.

The goal is to adjust W1 in a manner that reduce the error and W1 could be adjusted by .00438568.

However this might be too strong an adjustment.

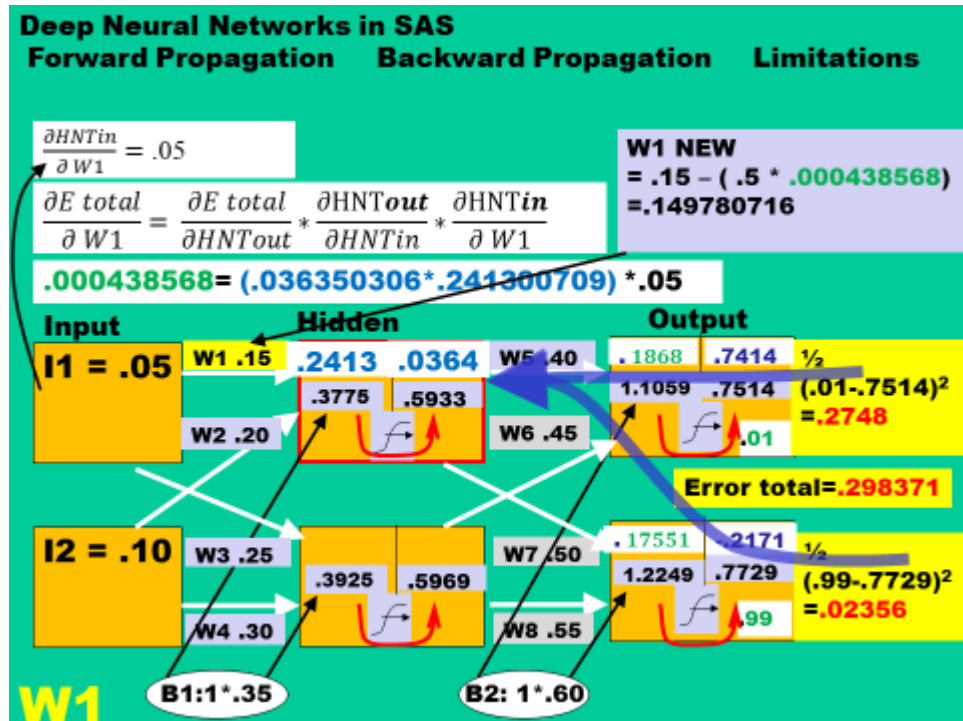


Figure 18

Adjusting by .00438568 might lead to overcorrection and wild oscillations. It is, generally, a better practice to take smaller steps toward the goal than to take large steps and overshoot the goal. Instead of adjusting by .00438568,

Enterprise Miner will apply a learning factor (here .5) to reduce the size of the adjustment. In this example, the algorithm will only make half the suggested correction in hopes of creating a more stable approach to our goal.

Note: this is a basic example of back prop and back prop is a hot area of research. Some newer algorithms will monitor changes in error as learning progresses and, dynamically, adjust the learning rate. These newer algorithms will “take bigger steps” towards the solution when possible.

The calculations for adjusting W2 to W4 are similar to those shown above and are left to the appendix.

RESTRICTED BOLTZMANN MACHINES (RBM):

The fact that back proposition involves the chain rule, and many multiplications, limited the depth of neural networks for several years. As networks got deeper the back prop algorithm had to multiply more and more terms. Generally those terms were close to zero and the repeated multiplication of small terms would drive the the result of the calculation down close to machine accuracy.

The formulas used above were calculating the gradient, the slope of the error shape, with respect to the different weights. When the formula drove the derivative of a weight to zero, the formula “told the algorithm” that there was no chance of improving the error by adjusting that weight. Applying the above algorithm to deep nets made for long training times and unstable answers. Nets were limited in depth until the application of the Restricted Boltzmann machine (RBM) to neural networks.

A Restricted Boltzmann Machine has the advantage of giving the network good starting weights that are not close to zero. A Restricted Boltzmann Machine avoids the problem of the vanishing gradient.

A RBM breaks a Deep Neural Network into many two-layer networks (see right).

The first of the two layers is called the input layer and the second layer, the one on the right, is called the hidden layer.

The two-layer network is trained so that the second layer simply reproduces the values in the first layer.

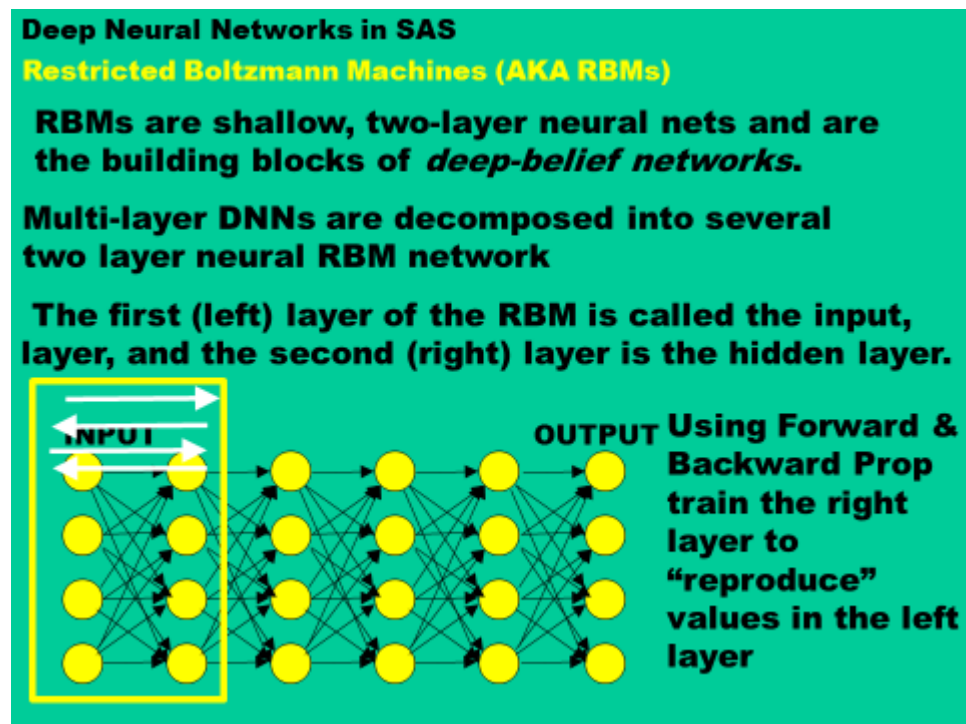


Figure 19

In figure 20 a reader can see the next step in the RBM. The process is to freeze weights between the input layer and hidden layer 1 and shift the RBM one layer to the right.

The RBM tries to make the hidden layer 3 reproduce the values in the hidden layer 2. This process continues until all the layers have been trained

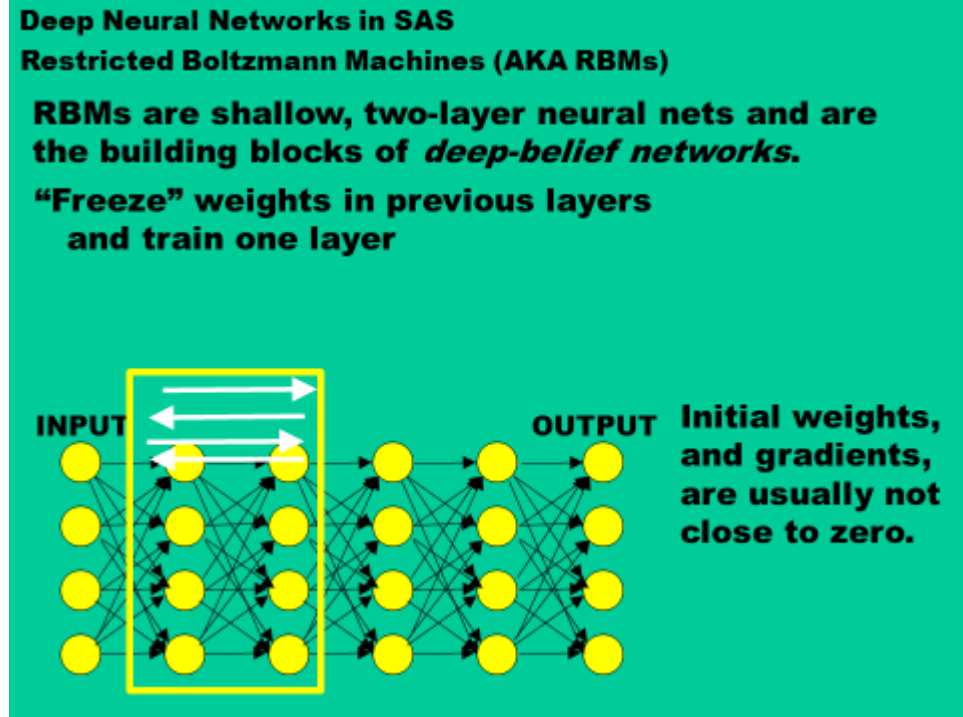


Figure 20

After all the layers have been trained, all their weights are unfrozen and the whole network is trained.

Early algorithms randomly assigning starting weights close to zero and this exacerbated the “vanishing gradient problem”.

Weights from the series of two-layer RBM training steps provide good, non-zero starting weights for training of the network.

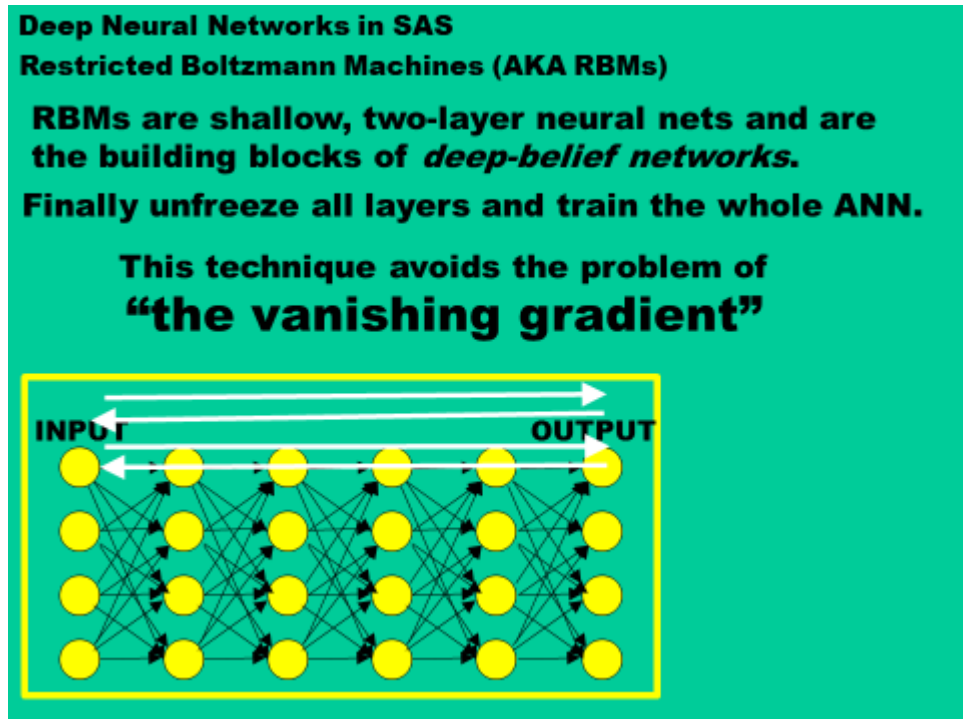


Figure 21

This technique avoids the vanishing gradient and has allowed researchers to use deeper and wider nets.

EXAMPLE: NEURAL NETWORKS ON HARD TO SEPARATE CLUSTERS

Figure 22 shows the one of the example problems that will be developed in this paper.

This example is from SAS online documentation.

A neural network will be used to separate these three groups.

The process will be to run PROC Neural in the SAS Display Manager. Proc Neural requires a DMDB catalog entry.

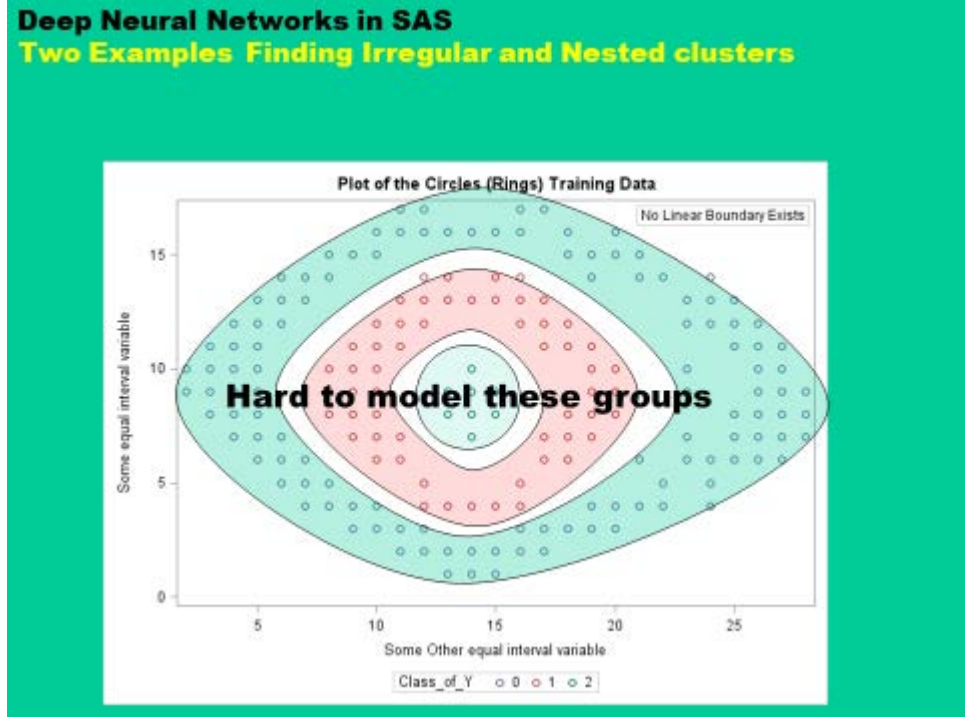


Figure 22

PROC DMDB creates a catalog entry containing metadata on the variables in the data set.

Think of PROC DMDB is adding information to what one sees when running a PROC Contents.

Proc Contents shows "data about the data".

Figure 24

Figure 25 shows the PROC Neural code with explanations for the statements.

SAS code to create this data set will be included in the appendix so an interested reader can conveniently run this example.

In the paper, we will skip to output to show how well this neural net performed.

```

Deep Neural Networks in SAS
Two Examples Finding Irregular and Nested clusters

PROC Neural data=Rings Data
              dmdbcat=DMDB_CatRings < Catalog
              random=789; Initialize weights randomly

input HORIZ VERT / level=interval id=i; Model
target Class of Y / id=o level=nominal;
hidden 3 / id=h; 1 layer and 3 nodes
prelim 5; Do preliminary weight training
train; Train now
score out=out outfit=fit; Scoring files: in and out
score data=sampsio.dmsring out=gridout;
title 'MLP with 3 Hidden Units';
run;

```

Figure 25

Figure 26 shows the results of the PROC Neural.

There were no misclassifications.

This is exciting performance on a highly non-linear data set.

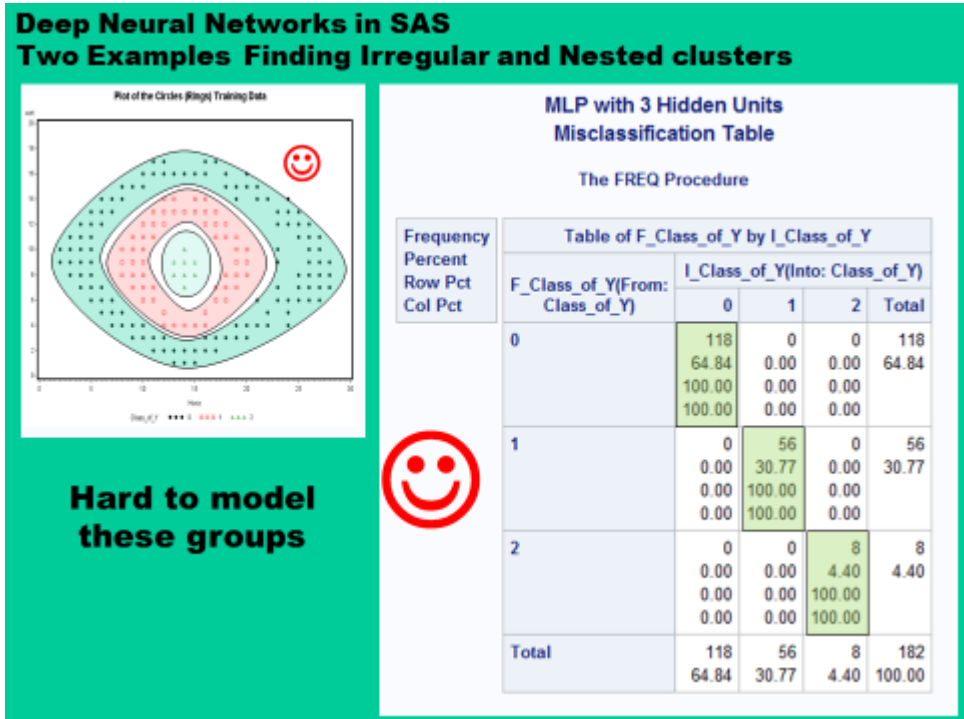


Figure 26

PROC Neural creates output data sets and a programmer that wants to build a PROC Neural into a larger project must understand the output.

The output from the PROC Neural will likely be input to some future steps.

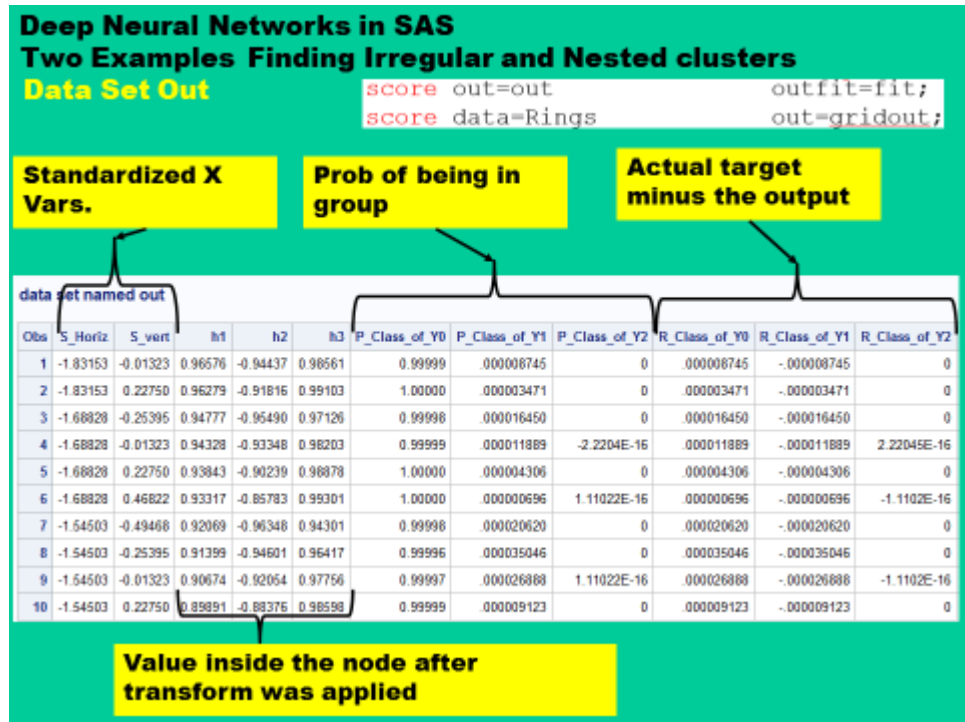


Figure 27

As this figure suggests, interpreting the contents of output from PROC Neural can be difficult.

E_ values depend on the method used in the Deep Neural Network.

After much research, I have not been able to find a definition of U_.

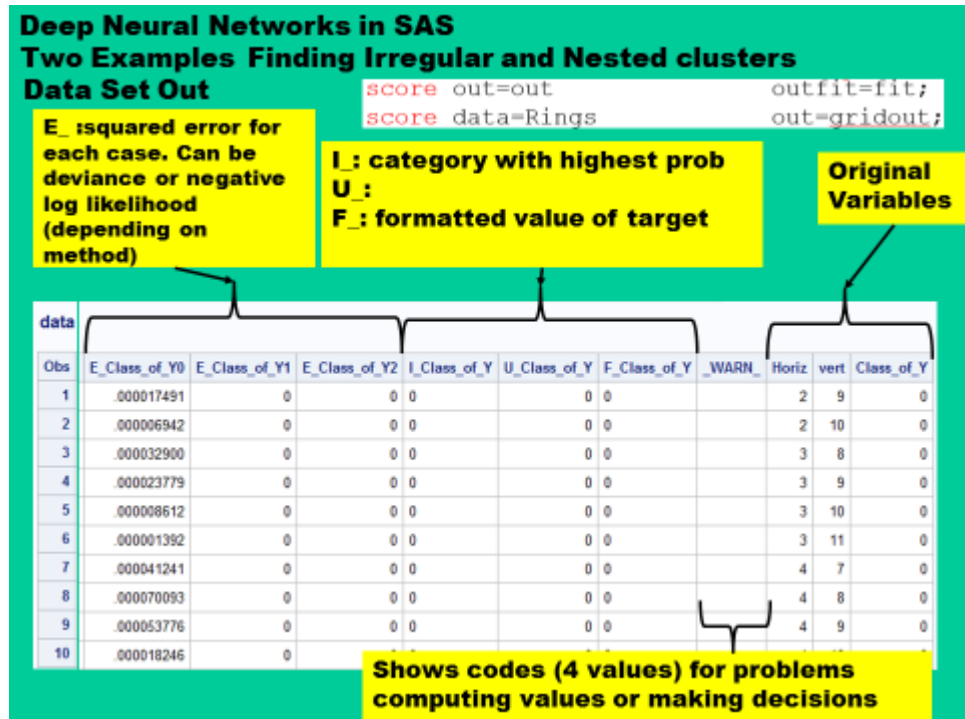


Figure 28

Rings was the input data set and was not changed.

Deep Neural Networks in SAS

Two Examples Finding Irregular and Nested clusters

Data Set RINGS

```
score out=out          outfit=fit;
score data=Rings       out=gridout;
```

data set named rings

Obs	Horiz	vert	Class_of_Y
1	2	9	0
2	2	10	0
3	3	8	0
4	3	9	0
5	3	10	0
6	3	11	0
7	4	7	0
8	4	8	0
9	4	9	0
10	4	10	0

Figure 29

EXAMPLE 2: PREDICTING LOAD DEFAULTS

This new example will try to predict loan defaults. This data set is shipped with SAS Enterprise Miner and an interested reader easily can run this code.

To the right, please see the use of PROC DMDB to create a catalog entry for use by PROC Neural.

Deep Neural Networks in SAS

Two Examples Finding defaults with a Deep Neural Network

Home Equity and Defaults

Obs	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
1	1	1100	25860	39025	HomeImp	Other	10.5	0	0	94.367	1	9	.
2	1	1300	70953	68490	HomeImp	Other	7.0	0	2	121.833	0	14	.
3	1	1500	13500	16700	HomeImp	Other	4.0	0	0	149.467	1	10	.
4	1	1500
5	0	1700	97800	112000	HomeImp	Office	3.0	0	0	93.333	0	14	.
6	1	1700	39548	40320	HomeImp	Other	9.0	0	0	101.466	1	8	37.1136
7	1	1800	48649	57037	HomeImp	Other	5.0	3	2	77.500	1	17	.
8	1	1800	28502	43034	HomeImp	Other	11.0	0	0	88.766	0	8	36.8849
9	1	2000	32760	46740	HomeImp	Other	3.0	0	2	216.933	1	12	.
10	1	2000	.	62250	HomeImp	Sales	15.0	0	0	115.800	0	13	.

#	Variable	Type	Len
1	BAD	Num	8
2	LOAN	Num	8
3	MORTDUE	Num	8
4	VALUE	Num	8
5	REASON	Char	7
6	JOB	Char	7
7	YOJ	Num	8
8	DEROG	Num	8
9	DELINQ	Num	8
10	CLAGE	Num	8
11	NINQ	Num	8
12	CLNO	Num	8
13	DEBTINC	Num	8

```
DATA HmEq_home_equity_Use;
SET DeepL.HmEq_home_equity;
RUN;QUIT;
```

```
PROC DMDB batch data=HmEq_home_equity_Use
out=DMDB_HmEq dmdbcat=DMDB_Cat_HmEq;
var /*bad*/ LOAN MORTDUE VALUE /*REASON JOB 7
YOJ DEROG DELINQ CLAGE NINQ CLNO DEBTINC;
class bad Job Reason;
target bad;
run;
```

Explorer

Name	Size	Type
Dmdb_cat_hmeq	21.0KB	Catalog
Dmdb_hmeq	640.0KB	Table
Hmeq_home_equity_use	704.0KB	Table

Figure 30

This PROC Neural call is fairly complicated and extends over two slides. The red boxes group similar types of commands.

PROC Neural allows a one to specify the number of CPUs to which s/he has access and to allow multithreading.

We are asking for three hidden layers. Hidden layer 1 has 36 nodes. Hidden layer 2 has 24 nodes and hidden layer 3 has two nodes.

Deep Neural Networks in SAS
Two Examples Finding defaults with a Deep Neural Network

```
options fulltimer; title "3 layer Neural Network";
PROC Neural data=HmEq home_equity_Use
dmdbcat=DMDB_Cat_HmEq
graph;
performance compile details cpucount=4 threads=yes;
/* ENTER CPU COUNT DO NOT EXCEED NUMBER OF PHYSICAL CORES */
/* DEFAULTS: ACT= TANH COMBINE= LINEAR */
/* IDS ARE USED AS LAYER INDICATORS - SEE FIGURE 6 */
/* INPUTS AND TARGETS SHOULD BE STANDARDIZED */

archi MLP hidden= 3;
hidden 36 / id= h1;
hidden 24 / id= h2;
hidden 2 / id= h3 act= linear;

input LOAN MORTDUE VALUE /*REASON JOB*/ YOJ DEROG DELINQ CLAGE NINQ
CLNO DEBTINC
/ id= i level= int std= std;
target bad / act= logistic id=t level= ordinal ;
/* BEFORE PRELIMINARY TRAINING WEIGHTS WILL BE RANDOM */
initial random= 123;
prelim 10 preiter= 30;
```

Data
← Catalog options

3 layers
36 nodes
24 nodes
2 nodes

Specify Model

Do preliminary training of weights

More code on next slide

Figure 31

This figure shows the coding of the RBMs. Each of the red boxes is an RBM and run in a sequence. The boxes will freeze and un-freeze appropriate hidden layers.

Deep Neural Networks in SAS
Two Examples Finding defaults with a Deep Neural Network

```
/*RBM-TRAIN LAYERS */
/*freeze i->h1*/
freeze h1->h2;
freeze h2->h3;
train technique= congra maxtime= 10000 maxiter= 10000;

freeze i->h1;
thaw h1->h2; /*train weights 1->2*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h1->h2;
thaw h2->h3; /*train the thirs layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

/* RETRAIN ALL LAYERS SIMULTANEOUSLY */
thaw i->h1;
thaw h1->h2;
thaw h2->h3;
train technique= congra maxtime= 10000 maxiter= 1000;

*code file= ' '/* ENTER SCORE CODE FILE PATH - SAME AS LINE 412 */

score out=HmEq_out outfit=HmEq_fit;
score data=HmEq_home_equity_Use out=HmEq_gridout;
title 'complex MLP ';
```

From Prev. Slide

Freeze hidden layers 1→3
Train input→hidden layer 1

Freeze & thaw layers
Train weights 1→2

Freeze & thaw layers
Train weights 2→3

Thaw all layers
Train all layers

Scoring: in and out

Figure 32

Since an interested reader can run this code, some output will be skipped and final results will be shown. Of the 1189 defaulters on the loan PROC Neural identified 309, or 26%.

Importantly, No alternative architectures were explored and the naively created node still correctly classified 85% of the people.

This example would be a good starting point for a reader wishing to play with neural networks

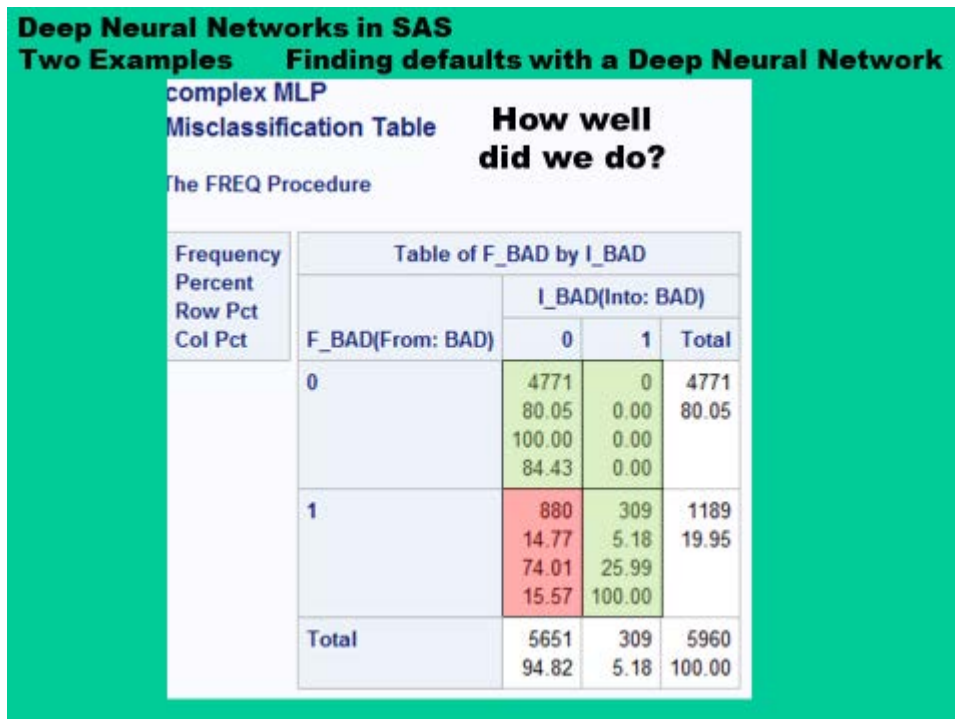


Figure 33

SUMMARY:

SAS PROC Neural is a very powerful modeling tool and analysts should consider some study of Deep Neural Networks.

ACKNOWLEDGMENTS:

Thanks to all the great people at SAS Tech Support.

CONTACT INFORMATION:

Your comments and questions are valued and encouraged. Contact the author at:
 Russ Lavery, Contractor
 Bryn Mawr, PA
 russ.lavery@verizon.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

***** **APPENDIX** *****

The appendix has three sections:

- 1) Code for the separate three groups example (the “rings” example)
- 2) Code for the predict delinquency example
- 3) All PowerPoint slides for the back propagation

1) code for the separate three groups example (the “rings” example)

Data Rings;

```
infile datalines trunccover firstobs=3;
input @1 Horiz 2. @5 vert 2. @10 Class_of_Y 1.;
datalines;
```

```
Horiz Vert Class_of_Y
```

```
1234567890
```

```
2 9 0
2 10 0
3 8 0
3 9 0
3 10 0
3 11 0
4 7 0
4 8 0
4 9 0
4 10 0
4 11 0
4 12 0
5 6 0
5 7 0
5 8 0
5 9 0
5 10 0
5 11 0
5 12 0
5 13 0
6 5 0
6 6 0
6 7 0
6 12 0
6 13 0
6 14 0
7 4 0
7 5 0
7 6 0
7 13 0
7 14 0
8 4 0
8 5 0
8 14 0
8 15 0
8 8 1
8 9 1
8 10 1
9 3 0
9 4 0
9 15 0
9 7 1
9 8 1
9 9 1
9 10 1
9 11 1
10 3 0
10 4 0
10 15 0
10 16 0
10 6 1
10 7 1
10 8 1
10 9 1
10 10 1
10 11 1
10 12 1
```


11 2 0
11 3 0
11 16 0
11 17 0
11 6 1
11 7 1
11 11 1
11 12 1
11 13 1
12 2 0
12 3 0
12 16 0
12 17 0
12 4 1
12 5 1
12 12 1
12 13 1
12 14 1
13 1 0
13 2 0
13 16 0
13 4 1
13 13 1
13 14 1
13 8 2
13 9 2
14 1 0
14 2 0
14 16 0
14 4 1
14 13 1
14 7 2
14 8 2
14 9 2
14 10 2
15 1 0
15 2 0
15 16 0
15 4 1
15 13 1
15 14 1
15 8 2
15 9 2
16 2 0
16 3 0
16 16 0
16 17 0
16 4 1
16 5 1
16 12 1
16 13 1
16 14 1
17 2 0
17 3 0
17 17 0
17 6 1
17 7 1
17 11 1
17 12 1
17 13 1
18 3 0
18 15 0

18 16 0
18 6 1
18 7 1
18 8 1
18 11 1
18 12 1
19 3 0
19 4 0
19 14 0
19 15 0
19 7 1
19 8 1
19 9 1
19 10 1
19 11 1
20 3 0
20 4 0
20 15 0
20 16 0
20 8 1
20 9 1
20 10 1
21 4 0
21 6 0
21 14 0
21 15 0
22 4 0
22 5 0
22 14 0
23 6 0
23 7 0
23 9 0
23 10 0
23 12 0
23 13 0
24 4 0
24 5 0
24 6 0
24 12 0
24 13 0
24 14 0
25 6 0
25 7 0
25 8 0
25 11 0
25 12 0
25 13 0
26 6 0
26 7 0
26 8 0
26 9 0
26 10 0
26 11 0
26 12 0
27 6 0
27 7 0
27 8 0
27 9 0
27 10 0
27 11 0
28 7 0
28 8 0

28 9 0

```
;  
run;
```

```
PROC DMDB batch data=Rings  
  out=DMDB_Rings  
  dmdbcat=DMDB_CatRings;  
  var Horiz vert ;  
  class Class_of_Y;  
  target Class_of_Y;  
run;
```

```
proc catalog catalog=work.DMDB_CatRings;  
contents;  
run;quit;
```

```
proc SGPlot data=Rings;
```

```
PROC SGPLOT DATA = Rings;  
  Scatter X = horiz Y = vert  
  /group=Class_of_Y;  
  YAXIS LABEL = 'Some equal interval variable' ;  
  XAXIS LABEL = 'Some Other equal interval variable';  
  TITLE 'Plot of the Circles (Rings) Training Data';  
  INSET 'No Linear Boundary Exists'/ POSITION = TOPRIGHT BORDER;  
RUN;
```

```
/*PROC GPlot data=Rings;*/  
/* plot Vert*Horiz=Class_of_Y /haxis=axis1 vaxis=axis2;*/  
/* symbol c=black i=none v=dot;*/  
/* symbol2 c=red i=none v=square;*/  
/* symbol3 c=green i=none v=triangle;*/  
/* axis1 c=black width=2.5 order=(0 to 30 by 5);*/  
/* axis2 c=black width=2.5 minor=none order=(0 to 20 by 2);*/  
/* title 'Plot of the Circles (Rings) Training Data';*/  
/*run;quit;*/
```

```
PROC Neural data=Rings  
  dmdbcat=DMDB_CatRings  
  random=789;  
  input HORIZ VERT / level=interval id=i;  
  target Class_of_Y / id=o level=nominal;  
  hidden 3 / id=h;  
  prelim 5;  
  train;  
  score out=out outfit=fit;  
  score data=Rings out=gridout;  
  title 'MLP with 3 Hidden Units';  
run;
```

```
proc print data=fit noobs label;  
  var _aic_ _ase_ _max_ _rfpe_ _misc_ _wrong_;  
  where _name_ = 'OVERALL';  
  title2 'Fits Statistics for the Training Data Set';  
run;
```

```
proc freq data=out;  
tables f_Class_of_Y*i_Class_of_Y;  
title2 'Misclassification Table';
```

```
run;
```

2) code for the predict delinquency example

```
title "Home Equity and Defaults";  
libname DeepL "E:\____Conferences_2016\data_2_USE";  
options nocenter;  
ods listing;  
proc contents data=DeepL.HmEq_home_equity varnum;  
run;
```

```
proc print data=DeepL.HmEq_home_equity (obs=10);  
run;
```

```
DATA HmEq_home_equity_Use;  
SET DeepL.HmEq_home_equity;  
RUN;QUIT;
```

```
PROC DMDB batch data=HmEq_home_equity_Use  
  out=DMDB_HmEq  
  dmdbcat=DMDB_Cat_HmEq;  
var /*bad*/ LOAN MORTDUE VALUE /*REASON JOB*/ YOJ DEROG DELINQ CLAGE NINQ CLNO DEBTINC;  
class bad Job Reason;  
target bad;  
run;
```

```
*** TRAIN 3 LAYER AUTOENCODER;  
*two kinds of statements - actions and options;  
options fullstimer;  
title "3 layer Neural Network";  
PROC Neural data=HmEq_home_equity_Use  
  dmdbcat=DMDB_Cat_HmEq  
  graph;  
  performance compile details cpucount=4 threads= yes; /* ENTER VALUE FOR CPU COUNT */  
  *nloptions MaxIter=10000; /* DO NOT EXCEED NUMBER OF PHYSICAL CORES  
*/  
  /* DEFAULTS: ACT= TANH COMBINE= LINEAR */  
  /* IDS ARE USED AS LAYER INDICATORS - SEE FIGURE 6 */  
  /* INPUTS AND TARGETS SHOULD BE STANDARDIZED */  
  /*we have 13 variables, so I will recode the number of nodes down from the numbers in the recognize  
numbers example*/  
  archi MLP hidden= 3;  
  hidden 36 / id= h1;  
  hidden 24 / id= h2;  
  hidden 2 / id= h3 act= linear;  
  input LOAN MORTDUE VALUE /*REASON JOB*/ YOJ DEROG DELINQ CLAGE NINQ CLNO DEBTINC  
    / id= i level= int std= std;  
  target bad / act= logistic id=t level= ordinal ;  
  /* BEFORE PRELIMINARY TRAINING WEIGHTS WILL BE RANDOM */  
  initial random= 123;  
  prelim 10 preiter= 10;  
  
  /* TRAIN LAYERS SEPARATELY */  
  /*freeze i->h1*/ /*train the first layer*/  
  freeze h1->h2;  
  freeze h2->h3;  
  train technique= congra maxtime= 10000 maxiter= 10000;  
  
  freeze i->h1;  
  thaw h1->h2; /*train the second layer*/  
  train technique= congra maxtime= 10000 maxiter= 10000;
```

```

freeze h1->h2;
thaw h2->h3; /*train the thirs layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

/* RETRAIN ALL LAYERS SIMULTANEOUSLY */
thaw i->h1;
thaw h1->h2;
thaw h2->h3;

train technique= congra maxtime= 10000 maxiter= 1000;

*code file= "; /* ENTER SCORE CODE FILE PATH - SAME AS LINE 412 */

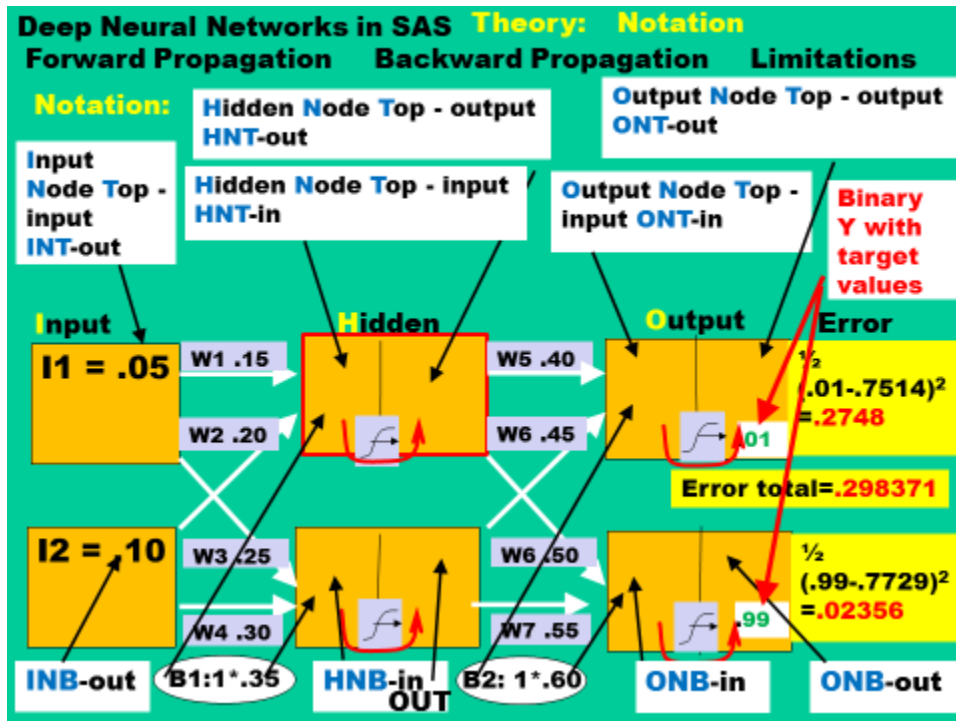
score out=HmEq_out outfit=HmEq_fit;
score data=HmEq_home_equity_Use out=HmEq_gridout;
title 'complex MLP';
run;

proc print data=HmEq_fit noobs label;
var _aic_ _ase_ _max_ _rfpe_ _misc_ _wrong_;
where _name_ = 'OVERALL';
title2 '3 layer Fits Statistics for the Training Data Set';
run;quit;

proc freq data=HmEq_out;
tables f_bad*i_bad;
title2 '3 LAYER Misclassification Table';
run;

```

3) all PowerPoint slides for the back propagation



Deep Neural Networks in SAS Theory: Notation

Forward Propagation Backward Propagation Limitations

Notation:

Hidden Node Top - output
HNT-out

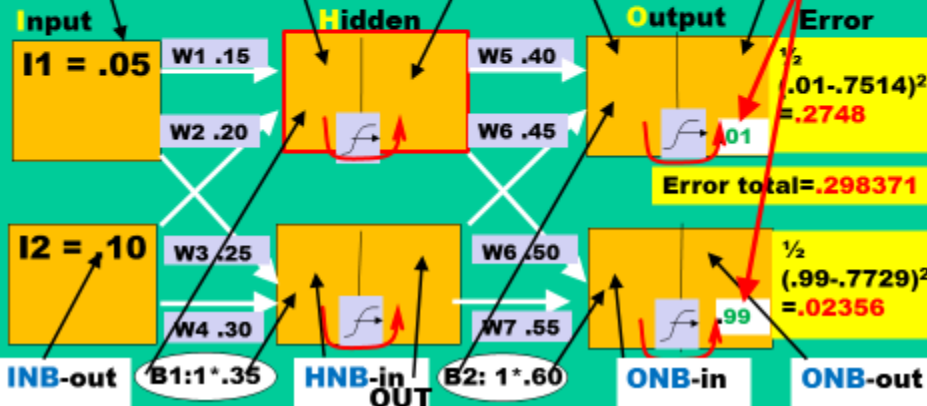
Output Node Top - output
ONT-out

Input Node Top - input
INT-out

Hidden Node Top - input
HNT-in

Output Node Top - input
ONT-in

Binary
Y with
target
values



Deep Neural Networks in SAS Theory:

Forward Propagation Backward Propagation Limitations

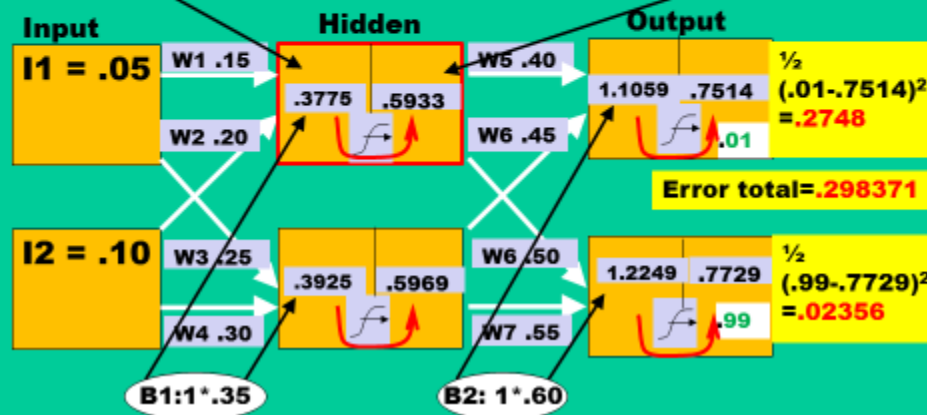
Forward Prop is simple

the transform inside the node is : $\frac{1}{(1 + e^{-x})} = \frac{1}{(1 + e^{-in})}$

$(.05 \cdot .15) + (.1 \cdot .20) + (1 \cdot .35) = .3775$

$1 / (1 + \exp(-.3775))$

Error total = .2748 + .02356 = .298371



Deep Neural Networks in SAS Theory:
Forward Propagation **Backward Propagation** Limitations

DNN is an area of hot and furious research

There is no one accepted DNN algorithm

People are combining algorithms to improve performance

One trick is, while doing backprop, to randomly remove, and then add back in, nodes.



Supposedly, this improves image recognition – especially for cats hiding behind things



Back Propagation for Weight 5

From:
A Step by Step Backpropagation Example
by Matt Mazur

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

Deep Neural Networks in SAS

Forward Propagation **Backward Propagation** Limitations

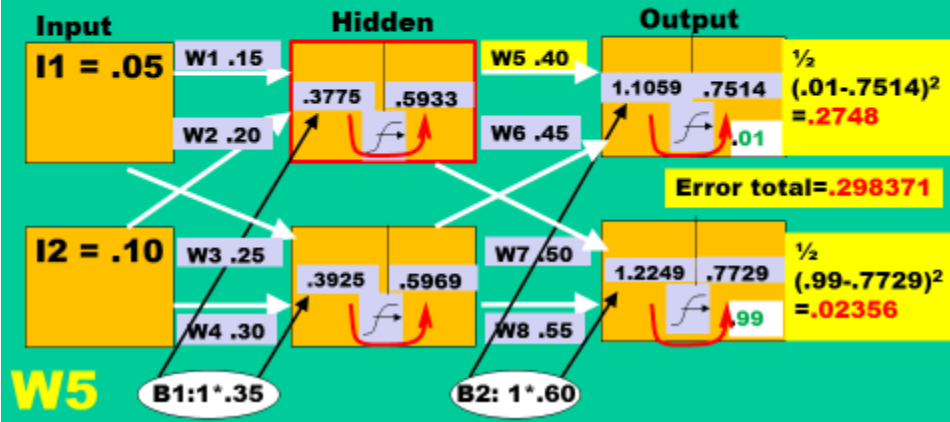
We need to “train”/change the weights to minimize TOTAL error

Lets look at W5 and start with ONT W5 only affects ONT

W5 affects ONT-IN&, **after going through the transform**, ONT-out and then error, so...

We need to compute three things →

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial ONT_{out}} * \frac{\partial ONT_{out}}{\partial ONT_{in}} * \frac{\partial ONT_{in}}{\partial w_5}$$



Deep Neural Networks in SAS

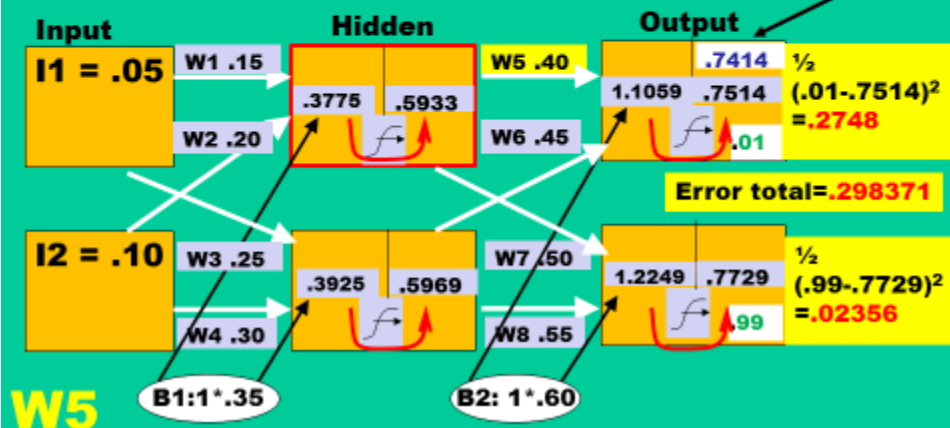
Forward Propagation **Backward Propagation** Limitations

$$E_{total} = \frac{1}{2} * (target_{4ONT} - ONT_{out})^2 + \frac{1}{2} * (target_{4ONB} - ONB_{out})^2$$

1: $\frac{\partial E_{total}}{\partial ONT_{out}}$

$$\frac{\partial E_{total}}{\partial ONT_{out}} = 2 * \frac{1}{2} * (target_{4ONT} - ONT_{out})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial ONT_{out}} = -1 * (target_{4ONT} - ONT_{out})^{2-1} = -(0.01 - .75136507)^1 = .74136507$$

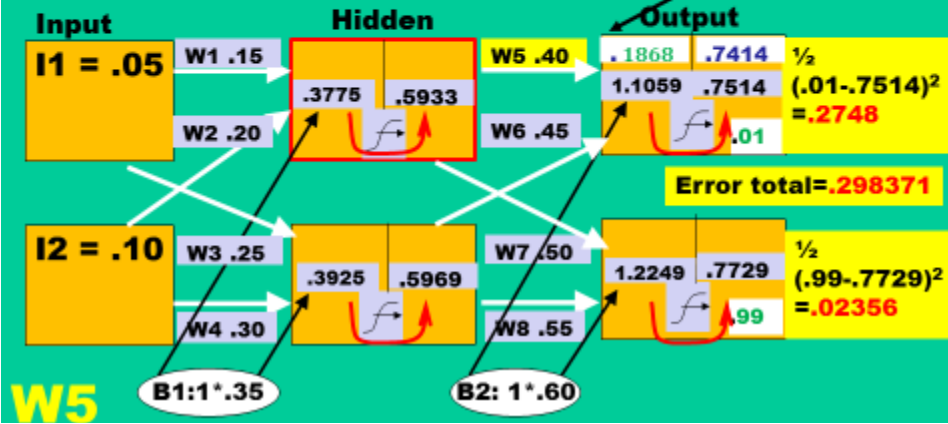


Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

the transform is: $\frac{1}{(1 + e^{-x})}$ the derivative is $x(1-x)$

2: $\frac{\partial ONout}{\partial ONTin}$

$\frac{\partial ONTout}{\partial ONTin} = ONTout(1 - ONTout) = 75136507(1 - .75136507) = .186815602$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

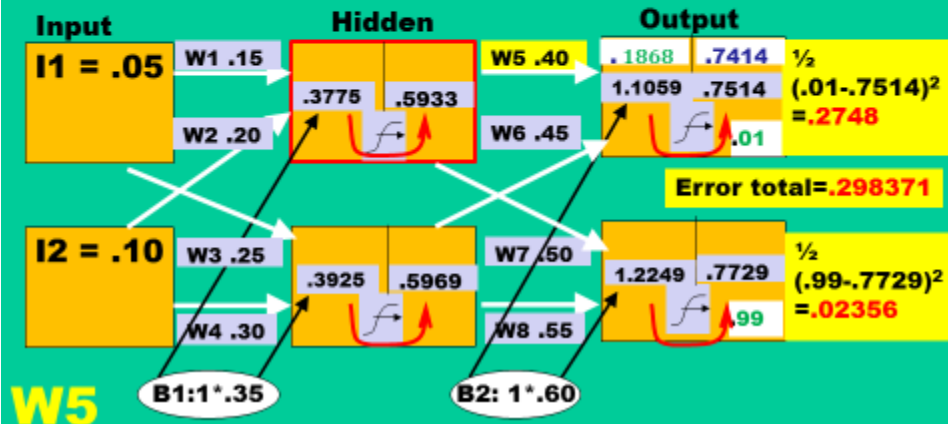
$ONTin = W5 * HNTout + W6 * HNBout + B2 * 1 = 1.1059$

#3 $\frac{\partial ONTin}{\partial w5} = 1 * HNTout * W5^{-1} + 0 + 0 = HNTout = 0.59327$

3: $\frac{\partial ONTin}{\partial w5}$

Remember $\frac{\partial Etotal}{\partial w5} = \frac{\partial Etotal}{\partial ONTout} * \frac{\partial ONTout}{\partial ONTin} * \frac{\partial ONTin}{\partial w5}$

$\frac{\partial Etotal}{\partial w5} = .74136507 * .186815602 * .0.59327 = 0.082167.$



Deep Neural Networks in SAS

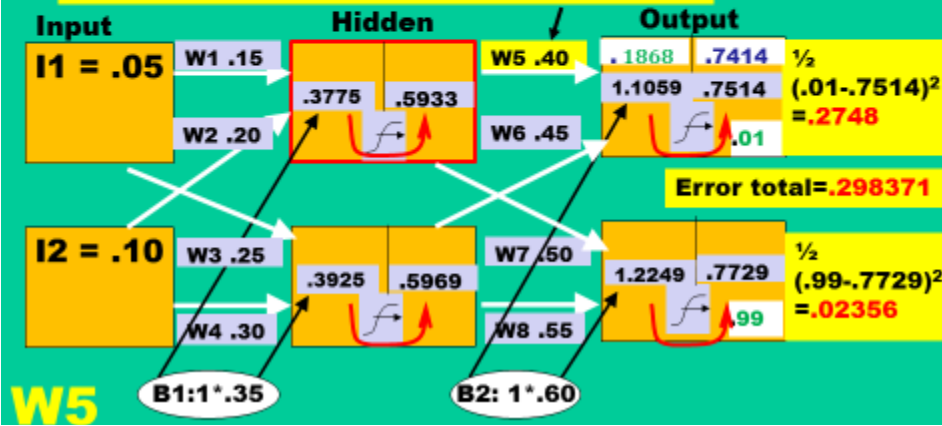
Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{total}}{\partial w_5} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out}) * HNT_{out}$$

$$\Delta = \text{delta} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out})$$

$$\frac{\partial E_{total}}{\partial w_5} = -\Delta * HNT_{out} = W_5 - \left(\text{LearnRate} * \frac{\partial E_{total}}{\partial w_5} \right)$$

$$W_{5\text{new}} = .4 - (.5 * .082167041) = .35891648$$



Back Propagation for Weight 6

Deep Neural Networks in SAS

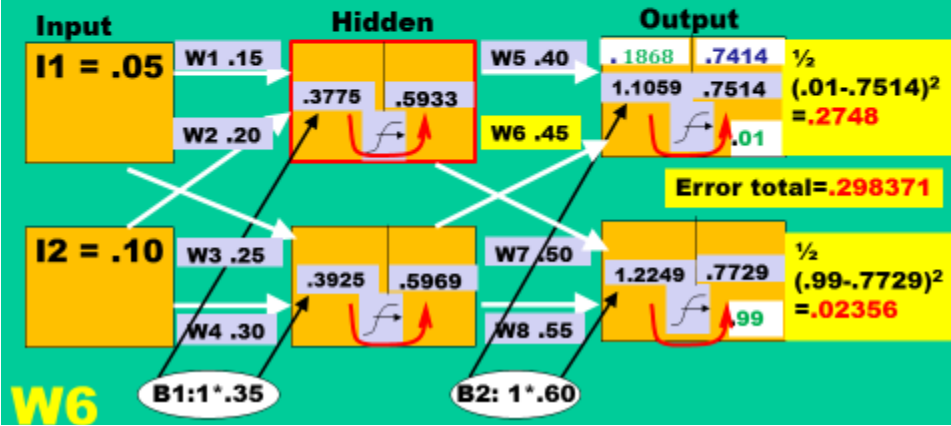
Forward Propagation **Backward Propagation** Limitations

We need to “train”/change the weights to minimize TOTAL error
 Lets look at W6 and start with ONT W6 only affects ONT

W6 affects ONT-IN&, after going through the transform, ONT-out and then error, so...

We need to compute three things →

$$\frac{\partial E_{total}}{\partial w_6} = \frac{\partial E_{total}}{\partial ONT_{out}} * \frac{\partial ONT_{out}}{\partial ONT_{in}} * \frac{\partial ONT_{in}}{\partial w_6}$$



Deep Neural Networks in SAS

Forward Propagation **Backward Propagation** Limitations

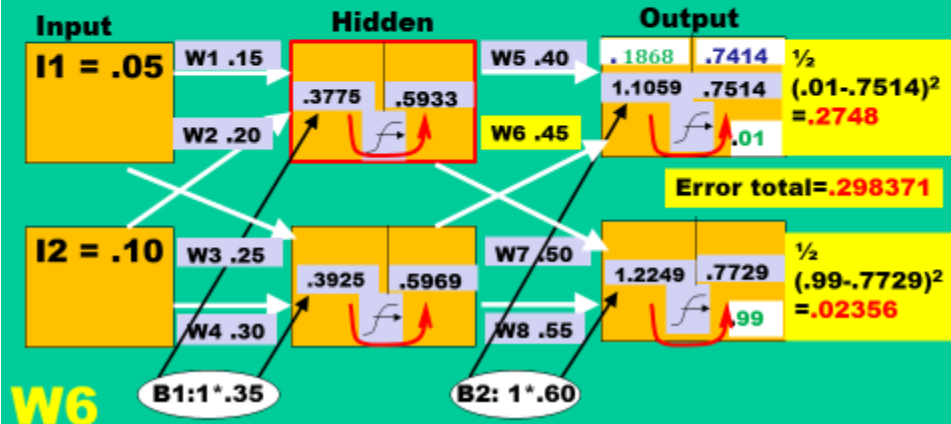
$$E_{total} = \frac{1}{2} * (target_{4ONT} - ONT_{out})^2 + \frac{1}{2} * (target_{4ONB} - ONB_{out})^2$$

1: $\frac{\partial E_{total}}{\partial ONT_{out}}$

$$\frac{\partial E_{total}}{\partial ONT_{out}} = 2 * \frac{1}{2} * (target_{4ONT} - ONT_{out})^{2-1} * -1 + 0$$

Same as W5

$$\frac{\partial E_{total}}{\partial ONT_{out}} = -1 * (target_{4ONT} - ONT_{out})^{2-1} = -(0.01 - .75136507)^1 = .74136507$$

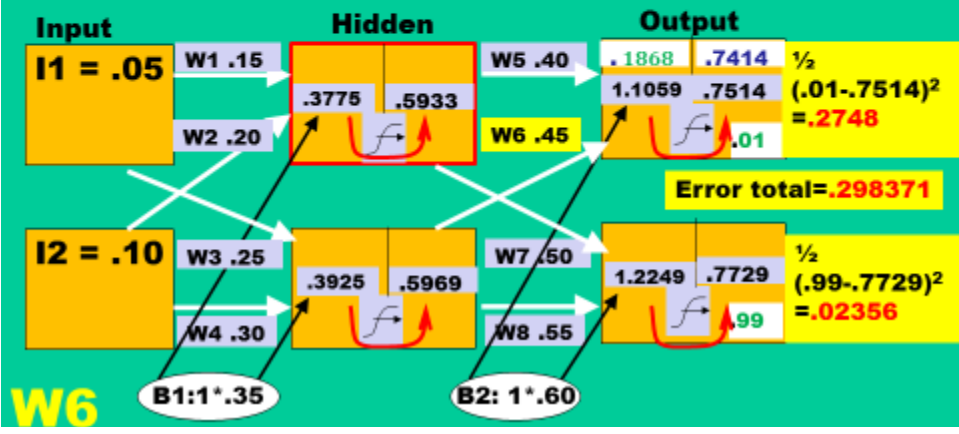


Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

2: $\frac{\partial ONout}{\partial ONTin}$

the transform is: $\frac{1}{(1 + e^{-x})}$ the derivitive is $x(1-x)$ **Same as W5**

$\frac{\partial ONTout}{\partial ONTin} = ONTout(1 - ONTout) = 75136507(1 - .75136507) = .186815602$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

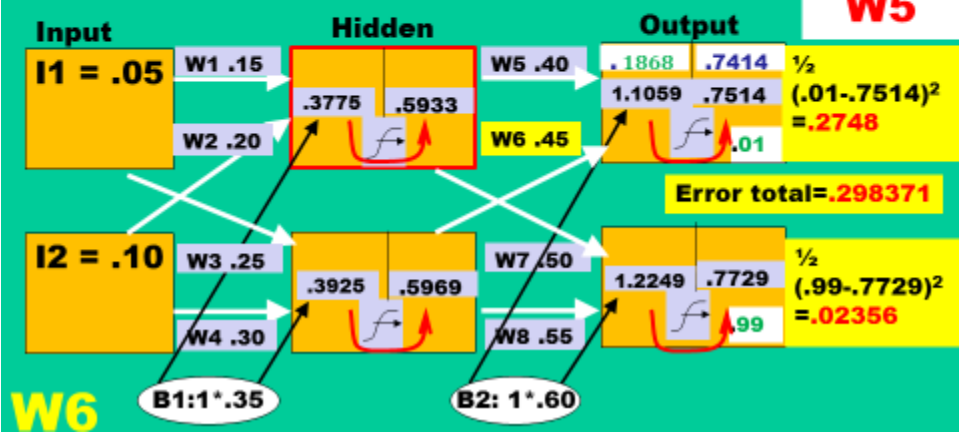
$ONTin = W5 * HNTout + W6 * HNBout + B2 * 1 = 1.1059$

#3 $\frac{\partial ONTin}{\partial W6} = 1 * HNBout * W6^{1-1} + 0 + 0 = HNBout = 0.59688437$ 3: $\frac{\partial ONTin}{\partial w6}$

Remember $\frac{\partial Etotal}{\partial w6} = \frac{\partial Etotal}{\partial ONTout} * \frac{\partial ONout}{\partial ONTin} * \frac{\partial ONTin}{\partial w6}$

$\frac{\partial Etotal}{\partial w6} = .74136507 * .186815602 * .0.596884378259767 = 0.08267.$

Same as W5



Deep Neural Networks in SAS

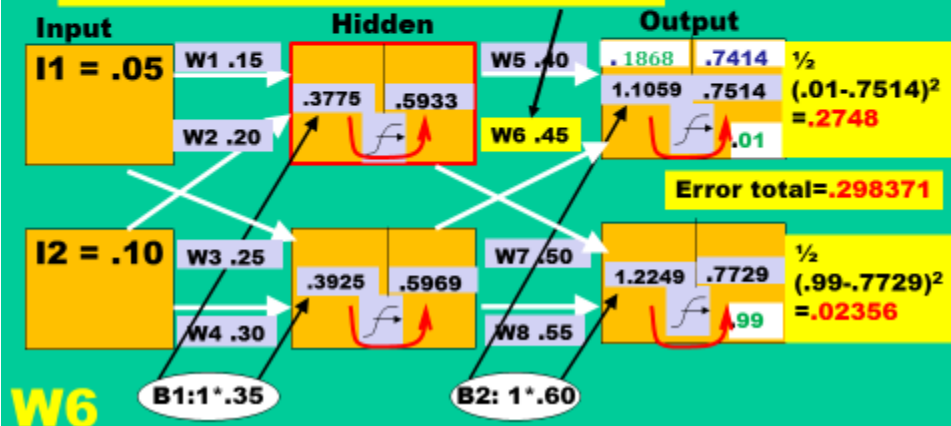
Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{total}}{\partial w_6} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out}) * HNB_{out}$$

$$\Delta = \text{delta} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out})$$

$$\frac{\partial E_{total}}{\partial w_6} = -\Delta * HNB_{out} = W_6 - \left(\text{LearnRate} * \frac{\partial E_{total}}{\partial w_6} \right)$$

$$W_{6\text{new}} = .45 - (.45 * .082167) = .40866$$



Back Propagation for Weight 7

Deep Neural Networks in SAS

Forward Propagation **Backward Propagation** Limitations

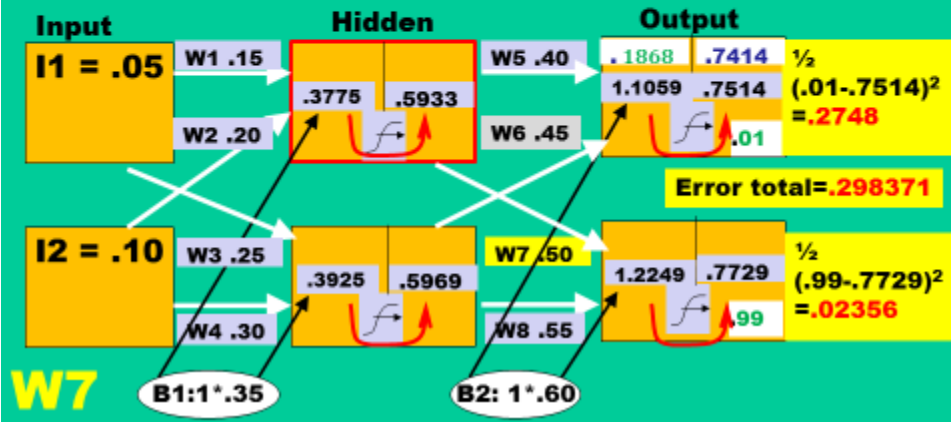
We need to “train”/change the weights to minimize TOTAL error

Lets look at W7 and start with ONB W7 only affects ONB

W7 affects ONB-IN&, **after going through the transform, ONB-out** and then error, so...

We need to compute three things →

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial ON_{Tout}} * \frac{\partial ON_{out}}{\partial ON_{Tn}} * \frac{\partial ON_{Tin}}{\partial w_7}$$



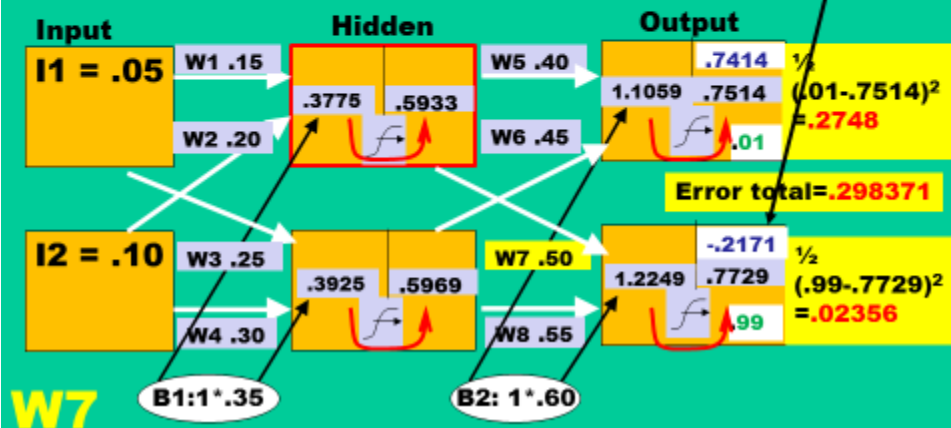
Deep Neural Networks in SAS

Forward Propagation **Backward Propagation** Limitations

$$E_{total} = \frac{1}{2} * (target_{ONB} - ON_{Bout})^2 + \frac{1}{2} * (target_{ONB} - ON_{Bout})^2 \quad 1: \frac{\partial E_{total}}{\partial ON_{Bout}}$$

$$\frac{\partial E_{total}}{\partial ON_{Bout}} = 2 * \frac{1}{2} * (target_{ONB} - ON_{Bout})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial ON_{Bout}} = -1 * (target_{ONB} - ON_{Bout})^{2-1} = -(.99 - .7729284)^1 = -.21707$$

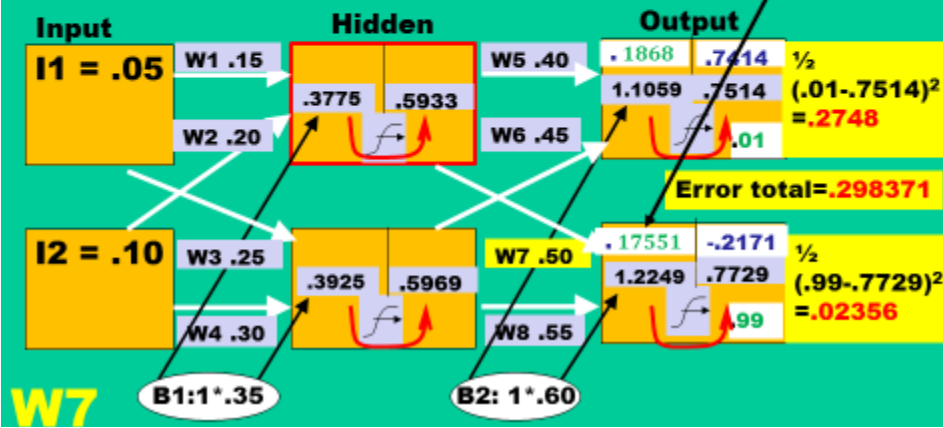


Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

the transform is: $1/(1 + e^{-x})$ the derivative is $x(1 - x)$

2: $\frac{\partial ONBout}{\partial ONBin}$

$\frac{\partial ONBout}{\partial ONBin} = ONBout(1 - ONBout) = .772928(1 - .772928) = .17551$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

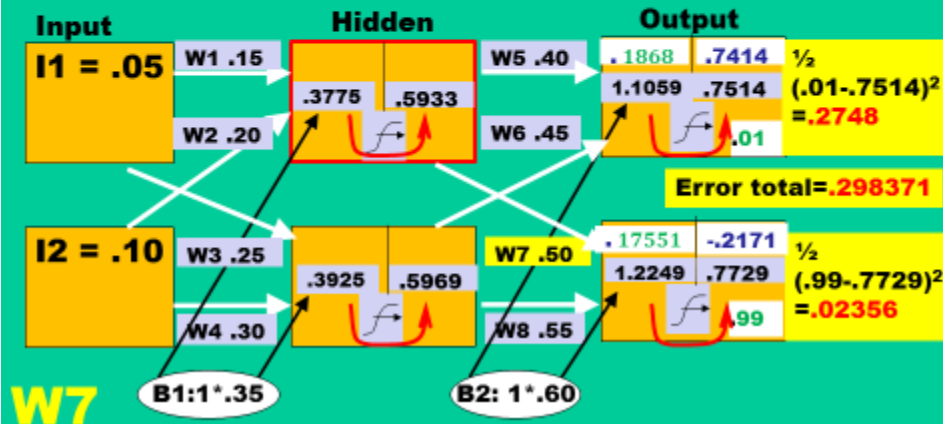
$ONBin = W7 * HNTout + W8 * HNBout + B2 * 1 = 1.2243$

#3 $\frac{\partial ONBin}{\partial W7} = 1 * HNTout * W7^{1-1} + 0 + 0 = HNTout = 0.59327$

3: $\frac{\partial ONBin}{\partial w7}$

Remember $\frac{\partial Etotal}{\partial w7} = \frac{\partial Etotal}{\partial ONTout} * \frac{\partial ONout}{\partial ONTin} * \frac{\partial ONTin}{\partial w7}$

$\frac{\partial Etotal}{\partial w7} = -.21707 * .17751 * .59327 = 0.082167.$



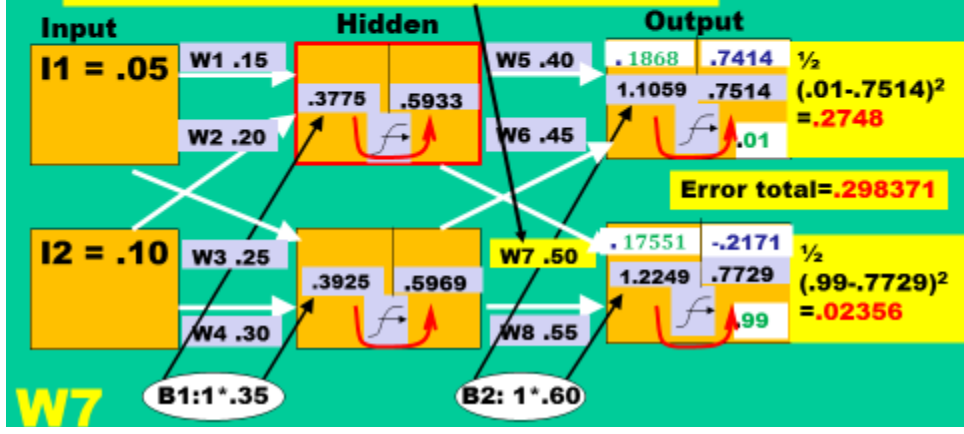
Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{total}}{\partial w_7} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out}) * HNT_{out}$$

$$\Delta = \text{delta} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out})$$

$$\frac{\partial E_{total}}{\partial w_7} = -\Delta * HNT_{out} \quad = W7 - (LearnRate * \frac{\partial E_{total}}{\partial w_7})$$

$$W7_{new} = .5 - (.5 * -.0226) = .5113$$



Back Propagation for Weight 8

Deep Neural Networks in SAS

Forward Propagation **Backward Propagation** Limitations

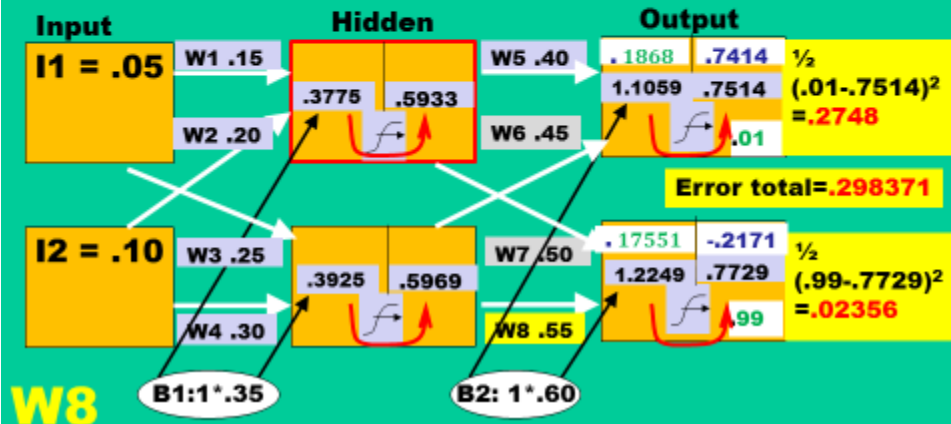
We need to “train”/change the weights to minimize TOTAL error

Lets look at W8 and start with ONB W8 only affects ONB

W8 affects ONB-IN&, **after going through the transform, ONB-out** and then error, so...

We need to compute three things →

$$\frac{\partial E_{total}}{\partial w_8} = \frac{\partial E_{total}}{\partial ONB_{out}} * \frac{\partial ONB_{out}}{\partial ONB_{in}} * \frac{\partial ONB_{in}}{\partial w_8}$$



Deep Neural Networks in SAS

Forward Propagation **Backward Propagation** Limitations

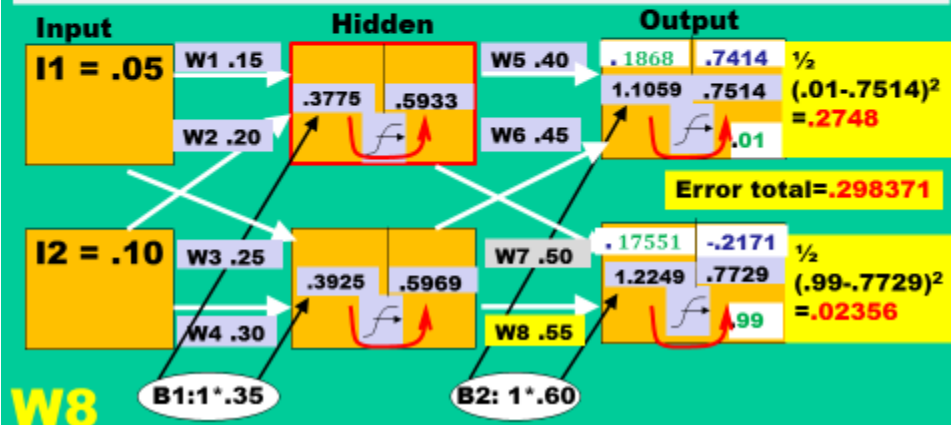
$$E_{total} = \frac{1}{2} * (target_{4ONB} - ONB_{out})^2 + \frac{1}{2} * (target_{4ONB} - ONB_{out})^2$$

$$1: \frac{\partial E_{total}}{\partial ONB_{out}}$$

$$\frac{\partial E_{total}}{\partial ONB_{out}} = 2 * \frac{1}{2} * (target_{4ONB} - ONB_{out})^{2-1} * -1 + 0$$

Same as W7

$$\frac{\partial E_{total}}{\partial ONB_{out}} = -1 * (target_{4ONB} - ONB_{out})^{2-1} = -(.99 - .7729284)^1 = -.21707$$



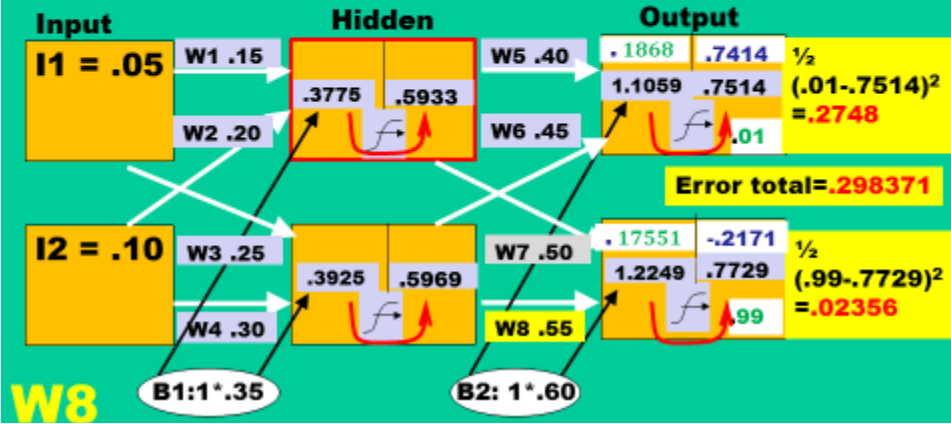
Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

the transform is: $1/(1 + e^{-x})$ the derivative is $x(1-x)$

2: $\frac{\partial ONBout}{\partial ONBin}$

Same as W7

$\frac{\partial ONBout}{\partial ONBin} = ONBout(1 - ONBout) = .772928(1 - .772928) = .17551$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

$ONBin = W7 * HNTout + W8 * HNBout + B2 * 1 = 1.2243$

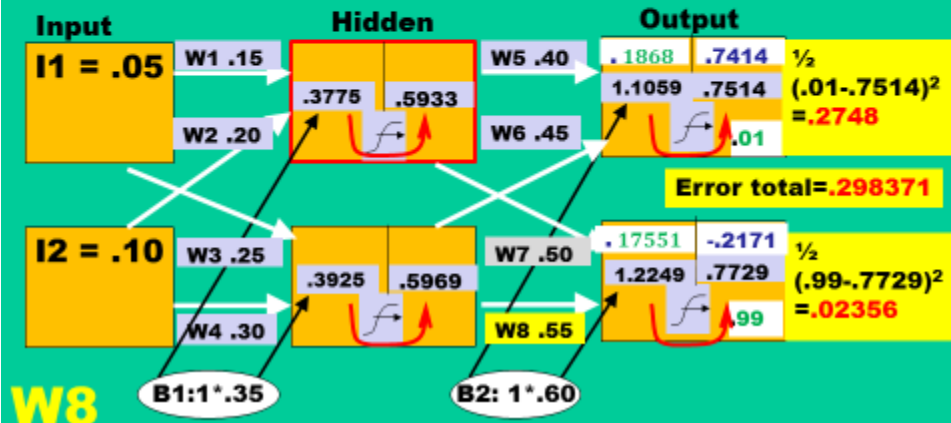
#3 $\frac{\partial ONBin}{\partial w8} = 1 * HNBout * w8^{-1} + 0 + 0 = HNBout = 0.59327$

Remember $\frac{\partial Etotal}{\partial w8} = \frac{\partial Etotal}{\partial ONTout} * \frac{\partial ONout}{\partial ONTin} * \frac{\partial ONTin}{\partial w8}$

3: $\frac{\partial ONBin}{\partial w8}$

Same as W7

$\frac{\partial Etotal}{\partial w8} = -.21707 * .17751 * .59327 = 0.082167$



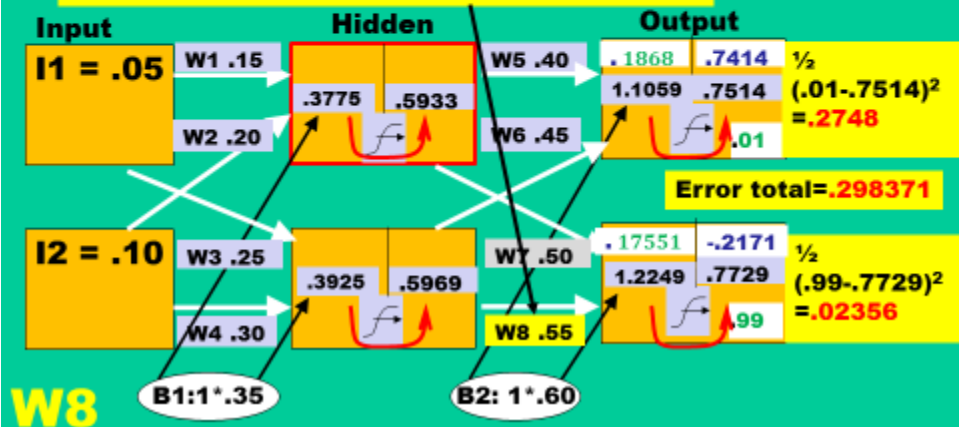
Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{total}}{\partial w_8} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out}) * HNT_{out}$$

$$\Delta = \text{delta} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out})$$

$$\frac{\partial E_{total}}{\partial w_8} = -\Delta * HNT_{out} = W_8 - (LearnRate * \frac{\partial E_{total}}{\partial w_5})$$

$$W_{8new} = .55 - (.5 * -.0226) = .56137$$



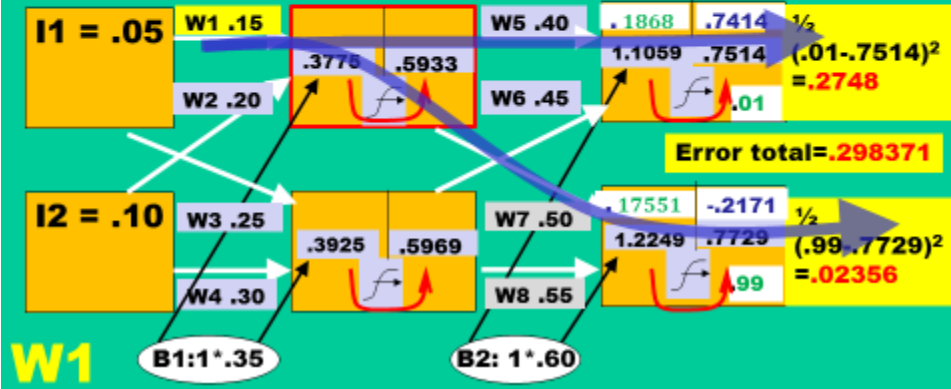
Back Propagation for Weight 1

Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

If you change w1 you change BOTH outputs

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial HNT_{out}} * \frac{\partial HNT_{out}}{\partial HNT_{in}} * \frac{\partial HNT_{in}}{\partial w_1}$$

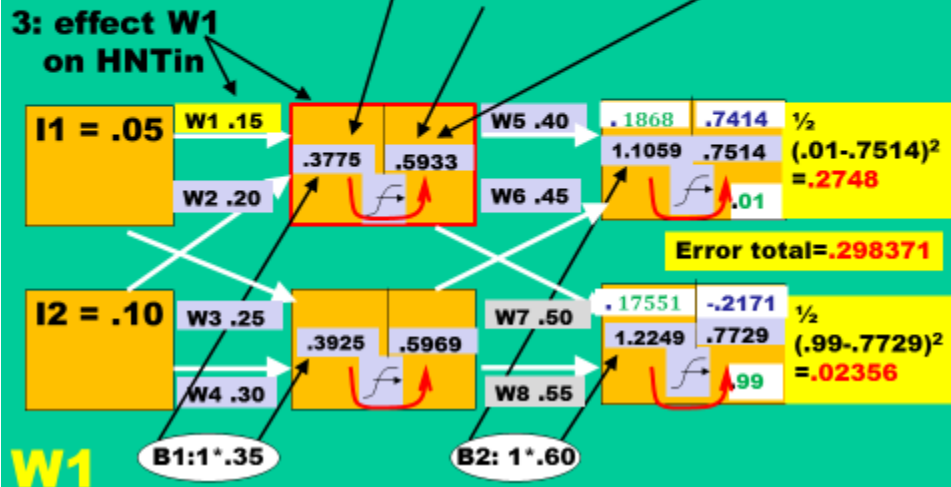
$$\frac{\partial E_{total}}{\partial HNT_{out}} = \frac{\partial E_{ONT}}{\partial HNT_{out}} * \frac{\partial E_{ONB}}{\partial HNT_{out}}$$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

3 steps again:

- 1: effect of output of HNTout on total error
- 2: effect of input to HNTin → HNT out



Deep Neural Networks in SAS

Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{ONT}}{\partial HNT_{out}} = \frac{\partial E_{ONT}}{\partial ONT_{in}} * \frac{\partial ONT_{in}}{\partial HNT_{out}}$$

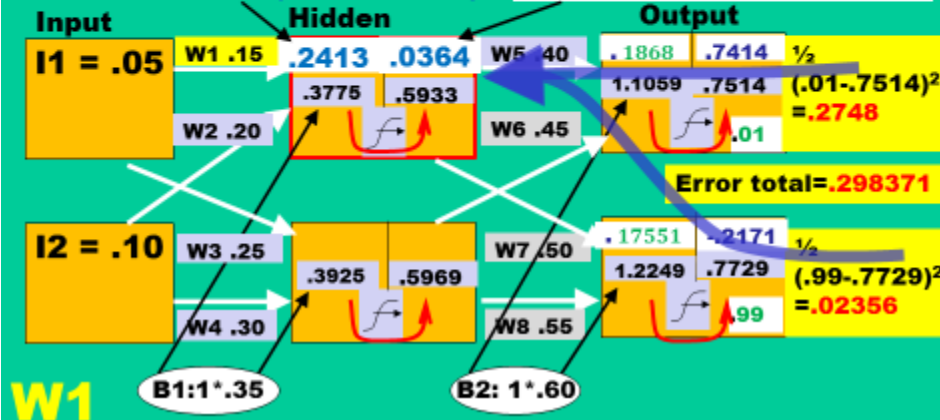
$$\frac{\partial E_{ONB}}{\partial HNT_{out}} = \frac{\partial E_{ONB}}{\partial ONB_{in}} * \frac{\partial ONB_{in}}{\partial HNB_{out}}$$

$$.241300700 = .59326999 * (1 - .59326999)$$

$$.055399425 = (.18615602 * .74136507) * .40$$

$$(.17551 * -.2171) * .50 = -.019049119$$

$$.055399425 - .019049119 = .036350306$$



Deep Neural Networks in SAS

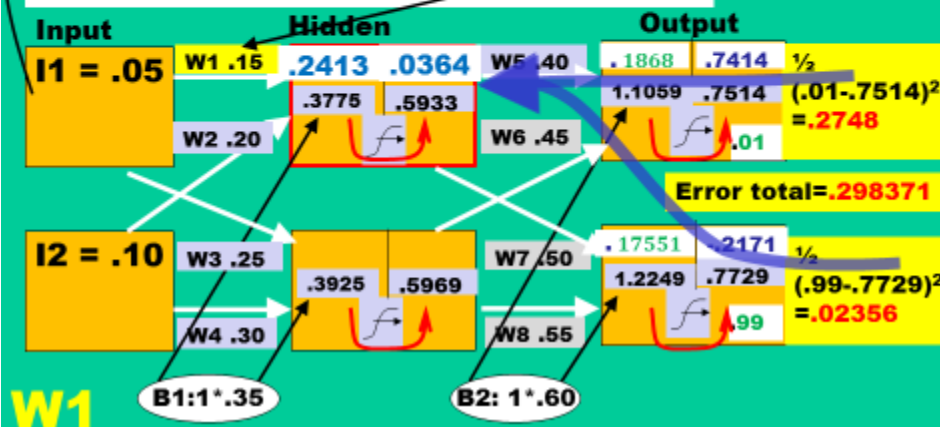
Forward Propagation Backward Propagation Limitations

$$\frac{\partial HNT_{in}}{\partial W_1} = .05$$

$$\frac{\partial E_{total}}{\partial W_1} = \frac{\partial E_{total}}{\partial HNT_{out}} * \frac{\partial HNT_{out}}{\partial HNT_{in}} * \frac{\partial HNT_{in}}{\partial W_1}$$

$$.000438568 = (.036350306 * .241300709) * .05$$

$$W_1 \text{ NEW} = .15 - (.5 * .000438568) = .149780716$$



Back Propagation for Weight 2

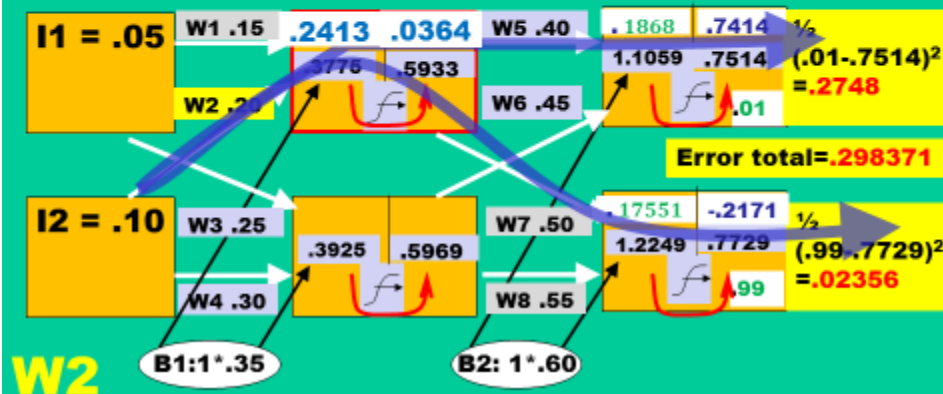
Deep Neural Networks in SAS

Forward Propagation Backward Propagation Limitations

If you change w2 you change BOTH outputs

$$\frac{\partial E_{total}}{\partial w_2} = \frac{\partial E_{total}}{\partial HNT_{out}} * \frac{\partial HNT_{out}}{\partial HNT_{in}} * \frac{\partial HNT_{in}}{\partial w_2}$$

$$\frac{\partial E_{total}}{\partial HNT_{out}} = \frac{\partial E_{ONT}}{\partial HNT_{out}} * \frac{\partial E_{ONB}}{\partial HNT_{out}}$$



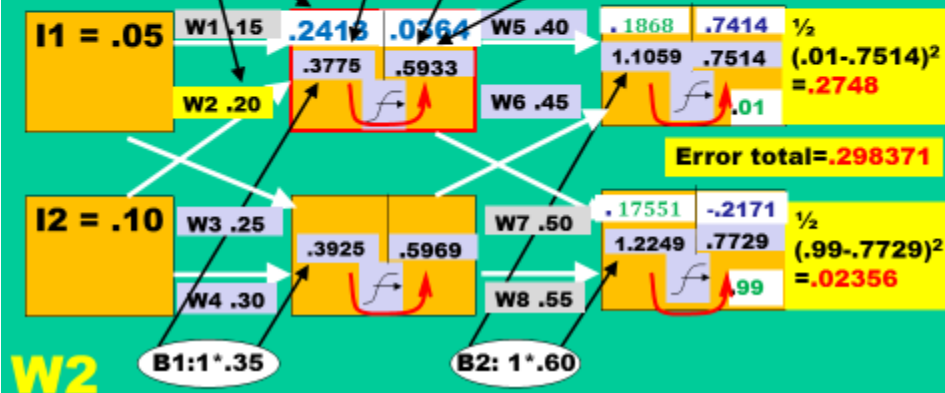
Deep Neural Networks in SAS
 Forward Propagation Backward Propagation Limitations

3 steps again:

1: effect of output of HNTout on total error

2: effect of input to HNTin → HNT out

3: effect W2 on HNTin



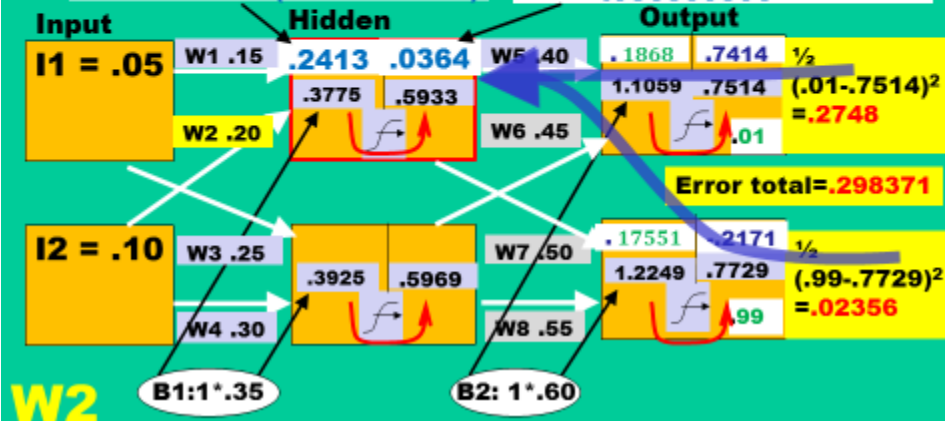
Deep Neural Networks in SAS
 Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{ONT}}{\partial HNTout} = \frac{\partial E_{ONT}}{\partial ONTin} * \frac{\partial ONTin}{\partial HNTout} = .055399425 = (.18615602 * .74136507) * .40$$

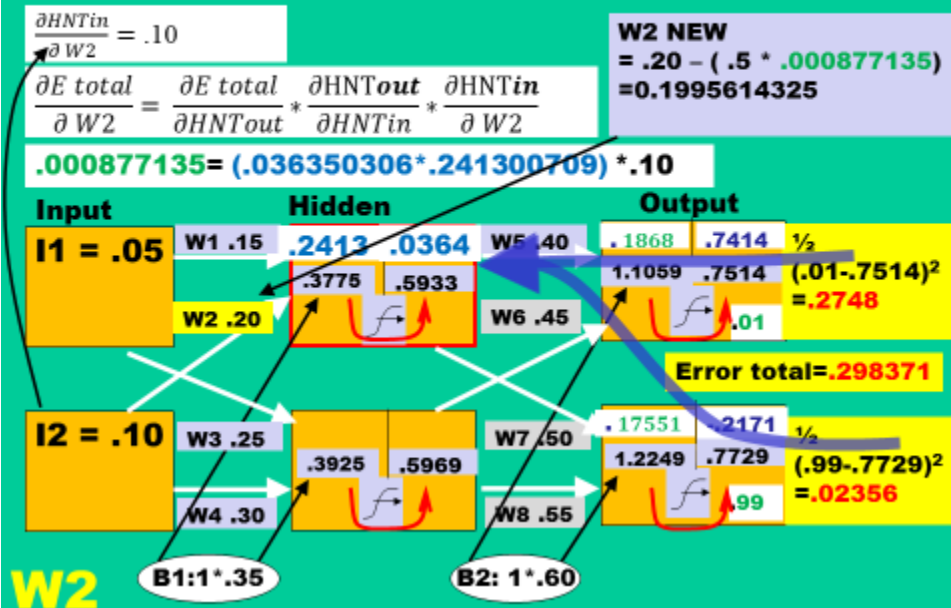
$$\frac{\partial E_{ONB}}{\partial HNTout} = \frac{\partial E_{ONB}}{\partial ONBin} * \frac{\partial ONBin}{\partial HNBout} = (.17551 * -.2171) * .50 = -.019049119$$

$$.241300700 = .59326999 * (1 - .59326999)$$

$$.055399425 - .019049119 = .036350306$$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations



Back Propagation for Weight 3

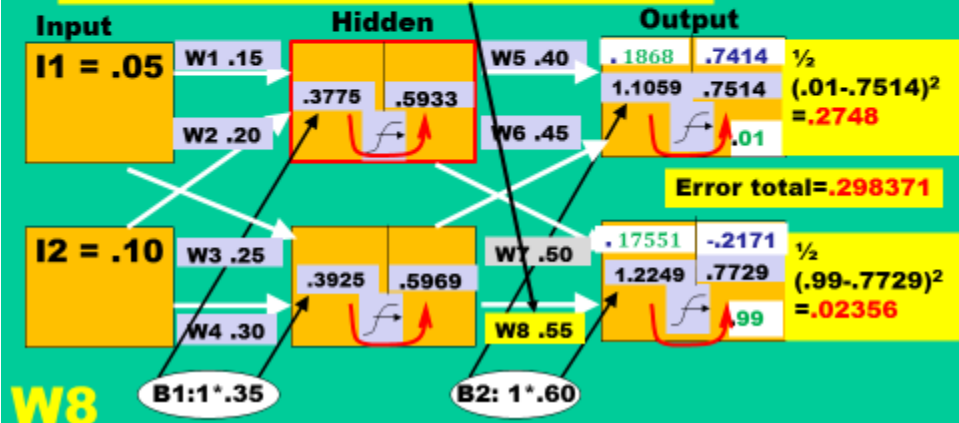
Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{total}}{\partial w_8} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out}) * HNT_{out}$$

$$\Delta = \text{delta} = -(ONT_{target} - ONT_{out}) * ONT_{out} * (1 - ONT_{out})$$

$$\frac{\partial E_{total}}{\partial w_8} = -\Delta * HNT_{out} = W_8 - \left(\text{LearnRate} * \frac{\partial E_{total}}{\partial w_5} \right)$$

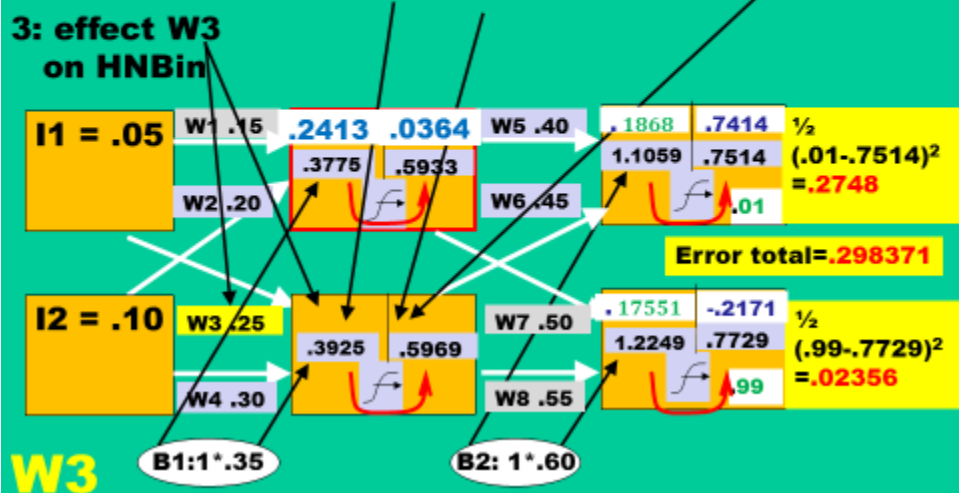
$$W_{8new} = .55 - (.5 * -.0226) = .56137$$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

3 steps again:

- 1: effect of output of HNbout on total error
- 2: effect of input to NHBin → HNB out



Deep Neural Networks in SAS

Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{ONT}}{\partial HNT_{out}} = \frac{\partial E_{ONT}}{\partial ONT_{in}} * \frac{\partial ONT_{in}}{\partial HNT_{out}}$$

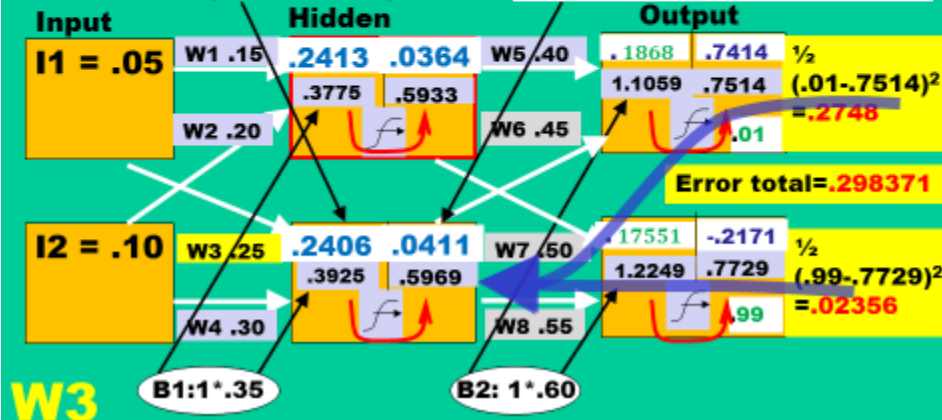
$$\frac{\partial E_{ONB}}{\partial HNT_{out}} = \frac{\partial E_{ONB}}{\partial ONB_{in}} * \frac{\partial ONB_{in}}{\partial HNB_{out}}$$

$$.062104307 = (.18615602 * .74136507) * .45$$

$$(.17551 * -.2171) * .55 = -.020956772$$

$$.240613491 = 0.596884 * (1 - 0.596884)$$

$$.062104307 - .020956772 = 0.041147535$$



Deep Neural Networks in SAS

Forward Propagation Backward Propagation Limitations

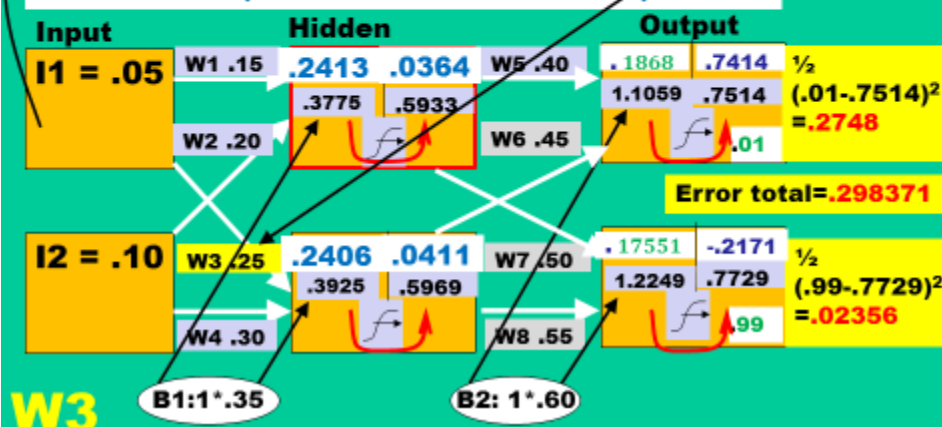
$$\partial HNT_{in} = W1 * .05 + W2 * .10$$

$$\frac{\partial HNT_{in}}{\partial W1} = .05$$

$$\frac{\partial E_{total}}{\partial W1} = \frac{\partial E_{total}}{\partial HNT_{out}} * \frac{\partial HNT_{out}}{\partial HNT_{in}} * \frac{\partial HNT_{in}}{\partial W1}$$

W3 NEW
 $= .25 - (.5 * 0.00049503)$
 $= 0.2497525$

$$0.00049503 = (0.041147535 * .240613491) * .05$$



Back Propagation for Weight 4

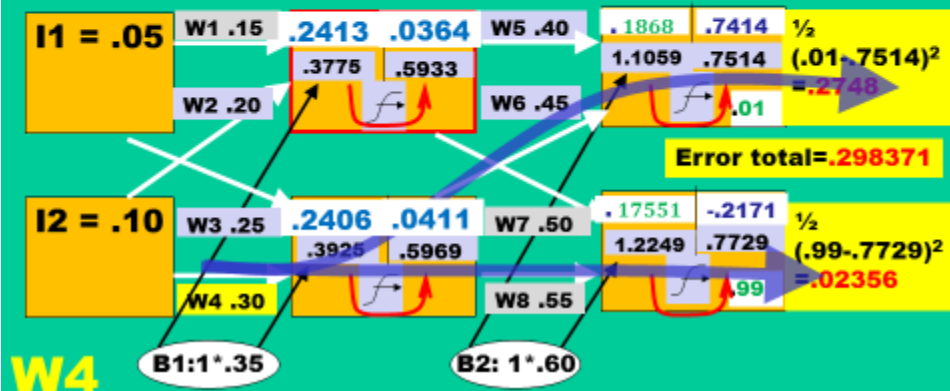
Deep Neural Networks in SAS

Forward Propagation Backward Propagation Limitations

If you change w4 you change BOTH outputs

$$\frac{\partial E_{total}}{\partial w_4} = \frac{\partial E_{total}}{\partial HNBout} * \frac{\partial HNBout}{\partial HNBin} * \frac{\partial HNBin}{\partial w_4}$$

$$\frac{\partial E_{total}}{\partial HNBout} = \frac{\partial E_{ONT}}{\partial HNBout} * \frac{\partial E_{ONB}}{\partial HNBout}$$



Deep Neural Networks in SAS
 Forward Propagation Backward Propagation Limitations

3 steps again:

1: effect of output of HNbout on total error

2: effect of input to NHBin → HNB out

3: effect W4 on HNBin



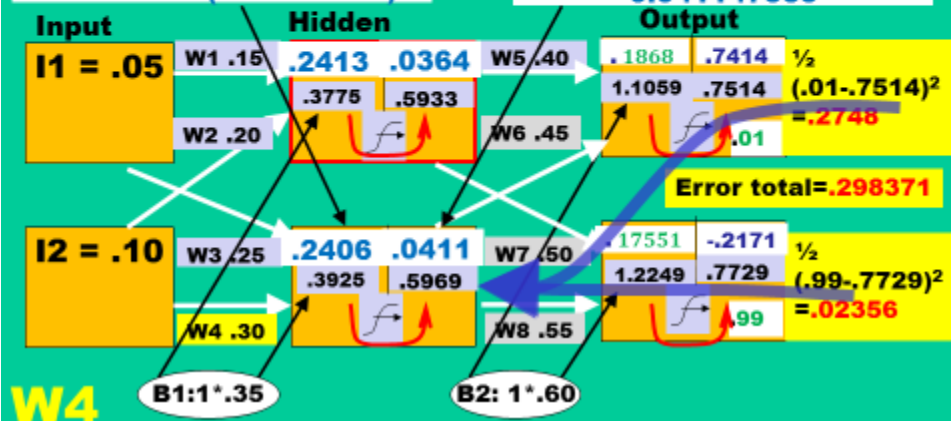
Deep Neural Networks in SAS
 Forward Propagation Backward Propagation Limitations

$$\frac{\partial E_{ONT}}{\partial HNTout} = \frac{\partial E_{ONT}}{\partial ONTin} * \frac{\partial ONTin}{\partial HNTout} = .062104307 = (.18615602 * .74136507) * .45$$

$$\frac{\partial E_{ONB}}{\partial HNTout} = \frac{\partial E_{ONB}}{\partial ONBin} * \frac{\partial ONBin}{\partial HNTout} = -.020956772 = (.17551 * -.2171) * .55$$

$$.240613491 = 0.596884 * (1 - 0.596884)$$

$$.062104307 - .020956772 = 0.041147535$$



Deep Neural Networks in SAS
Forward Propagation Backward Propagation Limitations

$$\partial HNTin = W1 * .05 + W2 * .10$$

$$\frac{\partial HNTin}{\partial W4} = .10$$

$$\frac{\partial E\ total}{\partial W4} = \frac{\partial E\ total}{\partial HNTout} * \frac{\partial HNTout}{\partial HNTin} * \frac{\partial HNTin}{\partial W4}$$

W4 NEW
 $= .30 - (.5 * 0.00099006)$
 $= .29950497$

$$0.000990065 = (0.041147535 * .240613491) * .10$$

